

Cloud 101

Building Servers as Cattle not Pets using Infrastructure as Code

Part 1
GUI Cloud Instance Creation

Part 2
Infrastructure as Code with Terraform

Skill Level: Beginner - No previous experience with software development or cloud required.

Course Length: 1.5 hours

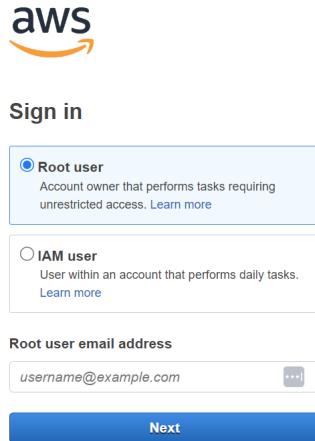
Prerequisites:

- Install Visual Studio Code
- Download Putty
- Bring your own AWS Account
- Github Repo Access

Presenter: David Thurm
Contact:
LinkedIn: [davidthurm](#)
Twitter: [@davidthurm](#)

Logging into the AWS Console

Console URL: <https://console.aws.amazon.com/>
Login Info: The AWS account you setup earlier.



Before We Begin

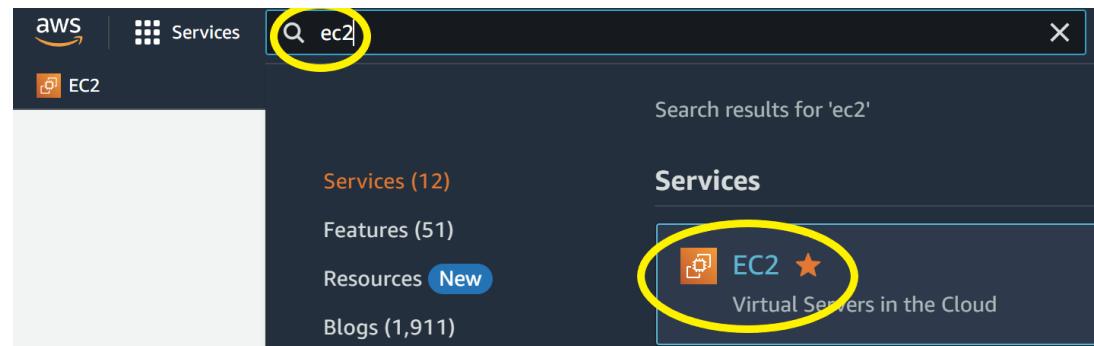
Step 1: Verify “US East (N. Virginia)” is selected

A screenshot of the AWS Console Home. The top navigation bar includes the AWS logo, a "Services" menu, a search bar, and a "Region" dropdown set to "N. Virginia". The main area shows "Console Home" and "Recently visited" sections. To the right is a "Regions" table with the following data:

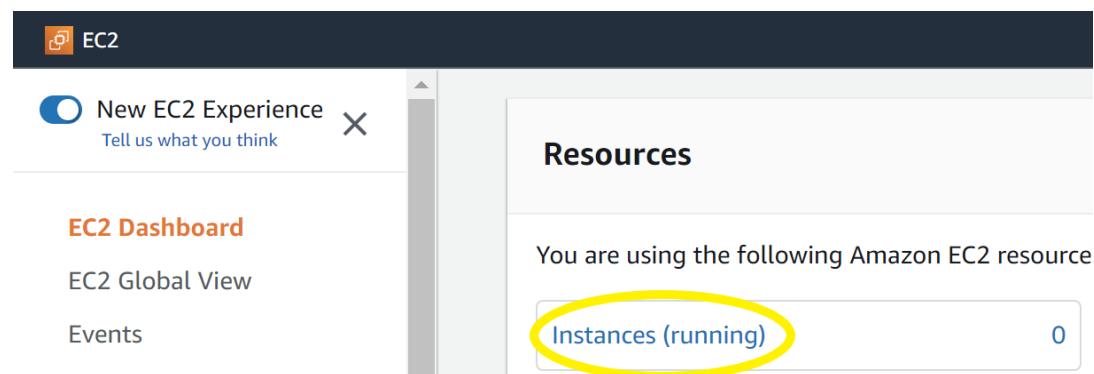
Region	Endpoint
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2

Create a new EC2 Instance

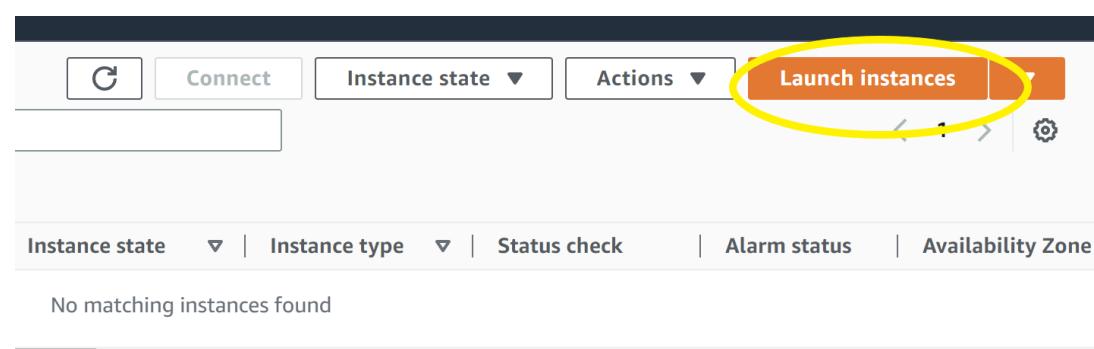
Step 1: Search > **EC2** > Select the EC2 Icon



Step 2: Select > Instances (running)



Step 3: Select > Launch Instances



Step 4: Name the Server > “Apache_Server”

EC2 > Instances > Launch an instance

Launch an instance Info

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. following the simple steps below.

Name and tags Info

Name

Apache_Server

Step 5: Choose your OS (ie Amazon Linux)

Search our full catalog including 1000s of application and OS images

Recents My AMIs Quick Start

Amazon Linux macOS Ubuntu Windows Red Hat S >

Amazon Machine Image (AMI)

Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type
ami-0dfcfc1ef8550277af (64-bit (x86)) / ami-0cd7323ab3e63805f (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs

Free tier eligible ▾

Description

Amazon Linux 2 Kernel 5.10 AMI 2.0.20230207.0 x86_64 HVM gp2

Architecture AMI ID

64-bit (x86) ami-0dfcfc1ef8550277af Verified provider

Not all operating systems are FREE.

Copy the AMI ID at the bottom to your notepad. We will use it later.

Step 6: Choose “**t2.micro**”

The screenshot shows a dropdown menu titled "Instance type" with "Info" link. The "t2.micro" option is highlighted with a yellow circle. To the right of "t2.micro", it says "Free tier eligible". Below "t2.micro", there is a list of details: Family: t2, 1 vCPU, 1 GiB Memory. It also lists On-Demand Windows pricing, On-Demand SUSE pricing, On-Demand RHEL pricing, and On-Demand Linux pricing.

Instance type	
t2.micro	Free tier eligible
Family: t2	1 vCPU
1 GiB Memory	
On-Demand Windows pricing:	0.0162 USD per Hour
On-Demand SUSE pricing:	0.0116 USD per Hour
On-Demand RHEL pricing:	0.0716 USD per Hour
On-Demand Linux pricing:	0.0116 USD per Hour

Key details are the CPU and Memory Available.

In this case above we see the server specs are barely has enough to run anything.

Free Tier Conditions:

- Operatings Systems bought in AWS Marketplace are **NOT** free.
- Only good for the first 12 months of your initial aws sign-up date.
- Some things have a trial period of less than the 12 months.

Create a New Key Pair

Step 1: Click “Create new key pair”

▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*

Select ▾

[Create new key pair](#)

Step 2: Name your key pair > “**Elyptic_Curve**”

Step 3: Choose Elyptic Curve “**ED25519**” (Notice there is no Windows support)

Step 4: Choose “**.ppk for use with putty.**”

Step 5: Click “**Create Key Pair**”

Create key pair

Key pairs allow you to connect to your instance securely.

Enter the name of the key pair below. When prompted, store the private key in a secure and accessible location on your computer. **You will need it later to connect to your instance.** [Learn more](#)

Key pair name

elliptic_curve

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type

RSA
RSA encrypted private and public key pair

ED25519
ED25519 encrypted private and public key pair (Not supported for Windows instances)

Private key file format

.pem
For use with OpenSSH

.ppk
For use with PuTTY

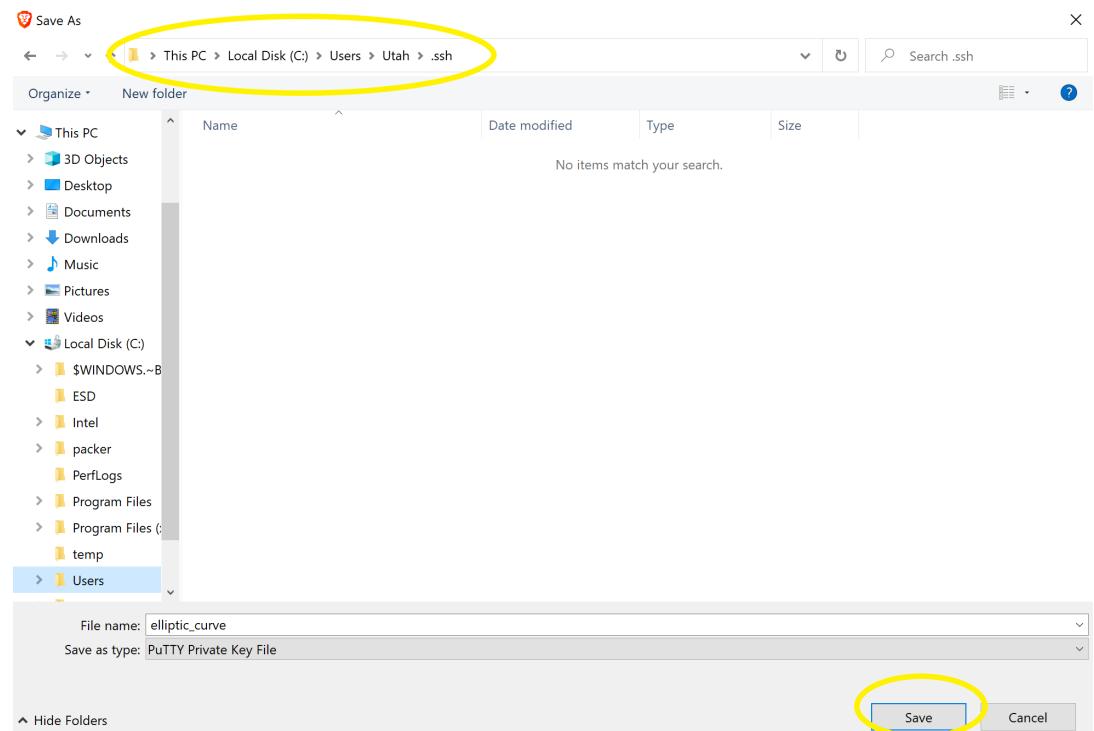
[Cancel](#) [Create key pair](#)

Step 6: Change directory to: C:\Users\your-username\

Step 7: Make a new folder named .ssh inside this folder

Step 8: Change directory to: C:\Users\your-username\.ssh

Step 9: Click "Save"



The use of the folder we created above is to provide a standardized location for all your ssh keys. As time goes on you will have several for different purposes.

We may need to move the file from the downloads to the folder we created.

Create a Security Group

Step 1: Click “**Create Security Group**”

Step 2: Checkmark “**Allow SSH traffic from**”.

Step 3: Dropdown and select “**My IP**”

▼ Network settings [Info](#) [Edit](#)

Network [Info](#)
vpc [REDACTED]

Subnet [Info](#)
No preference (Default subnet in any availability zone)

Auto-assign public IP [Info](#)
Enable

Firewall (security groups) [Info](#)
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

Create security group Select existing security group

We'll create a new security group called '**launch-wizard-1**' with the following rules:

Allow SSH traffic from
Helps you connect to your instance

Allow HTTPS traffic from the internet
To set up an endpoint, for example when

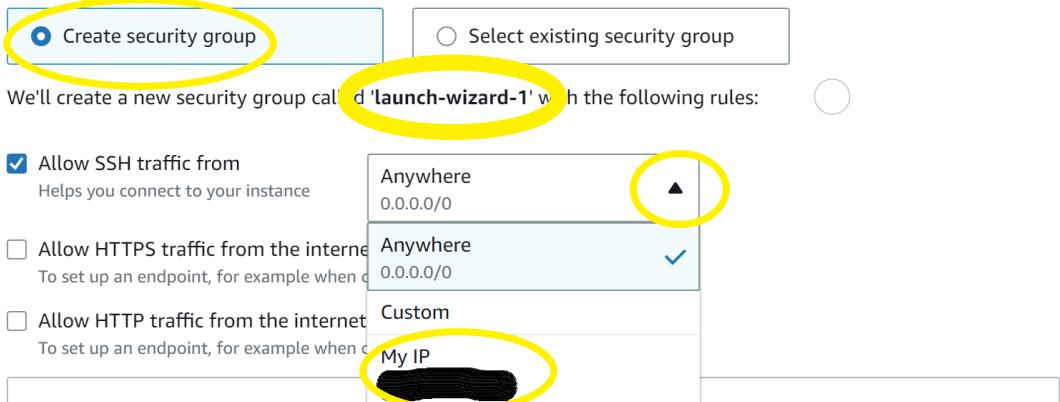
Allow HTTP traffic from the internet
To set up an endpoint, for example when

Anywhere
0.0.0.0/0

Anywhere
0.0.0.0/0

Custom

My IP



Step 4: Write down the Security Group name above.

In this case it is “**launch-wizard-1**”.

Configure Storage

Step 1: Choose “**30**” GB of hard disk space.

Step 2: Dropdown and choose “**GP3**” Storage Type.

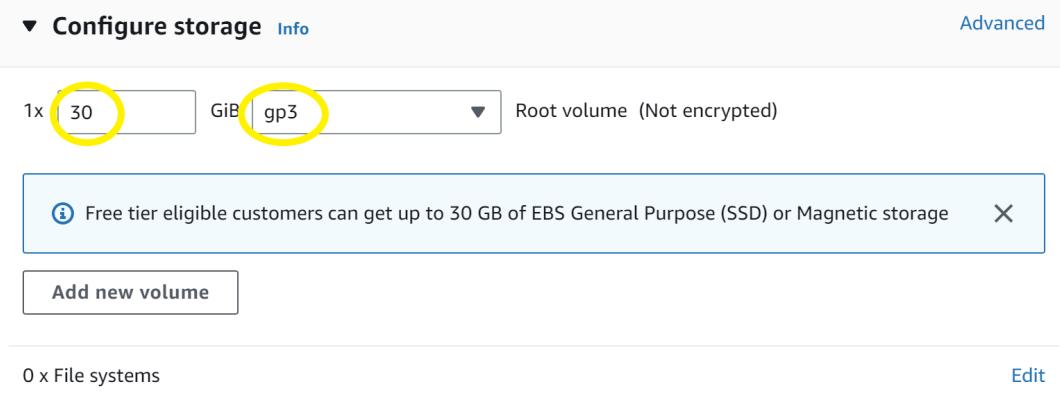
▼ Configure storage [Info](#) [Advanced](#)

1x GiB Root volume (Not encrypted)

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage [X](#)

[Add new volume](#)

0 x File systems [Edit](#)



Launch the Instance

Step 1: Verify information is correct

Step 2: Click “**Launch Instance**”

▼ Summary

Number of instances [Info](#)
1

Software Image (AMI)
Amazon Linux 2023 AMI 2023.0.2...[read more](#)
ami-02396cdd13e9a1257

Virtual server type (instance type)
t2.micro

Firewall (security group)
New security group

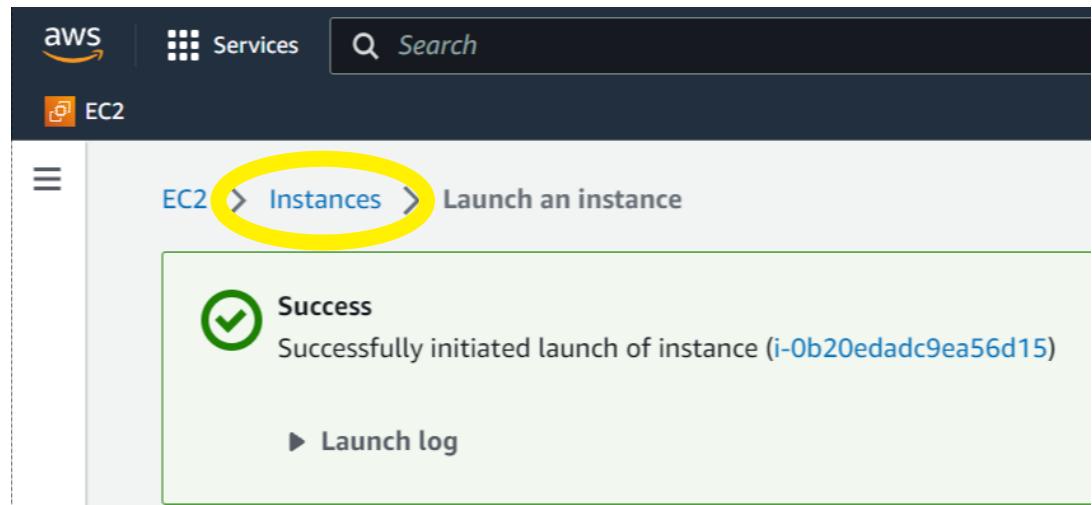
Storage (volumes)
1 volume(s) - 30 GiB

[Cancel](#) **Launch instance** [Review commands](#)



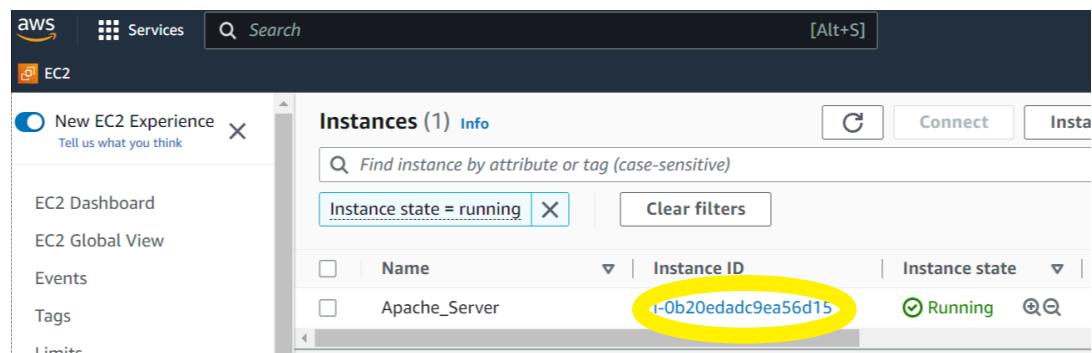
Gathering Server Instance IP Address

Step 1: Click “**Instances**”



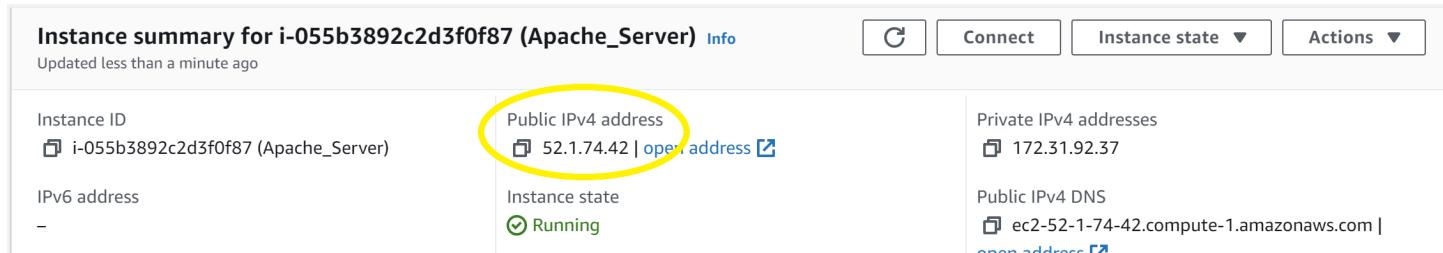
Step 2: Click “**Instance ID**” (Next to the Name you gave the server.)

In the image below it would be the ID “i-0b20edadc9ea56d15”.



Step 1: Copy the **Public IP** address to your Notepad.

We will use this later.



Configure Putty

Step 1: Download Putty

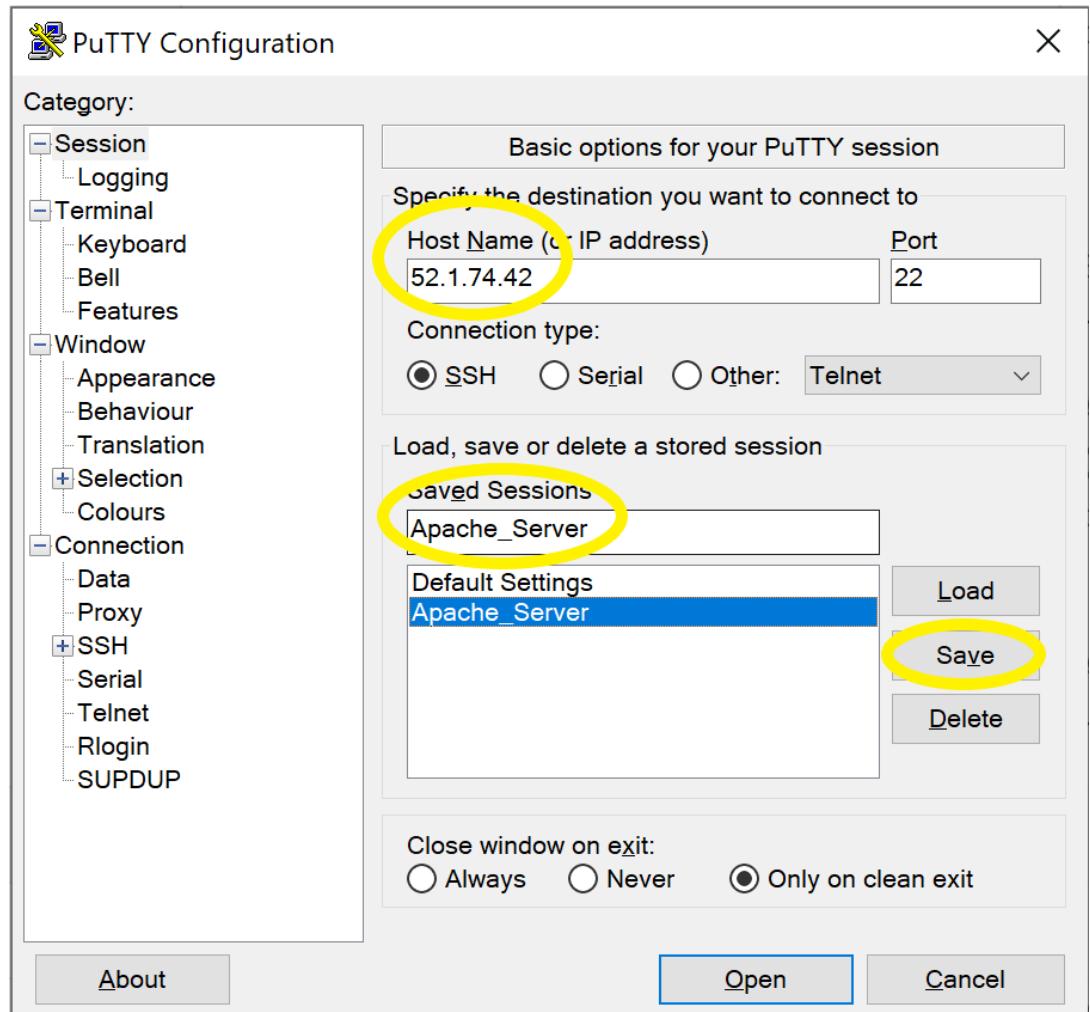
<https://the.earth.li/~sgtatham/putty/latest/w64/putty.exe>

Step 2: Run Putty by double clicking the icon.

Step 3: Copy the **IP Address** we saved into the Host Field of Putty.

Step 4: Enter the **Name** of the server in the Saved Sessions box.

Step 5: Click Save



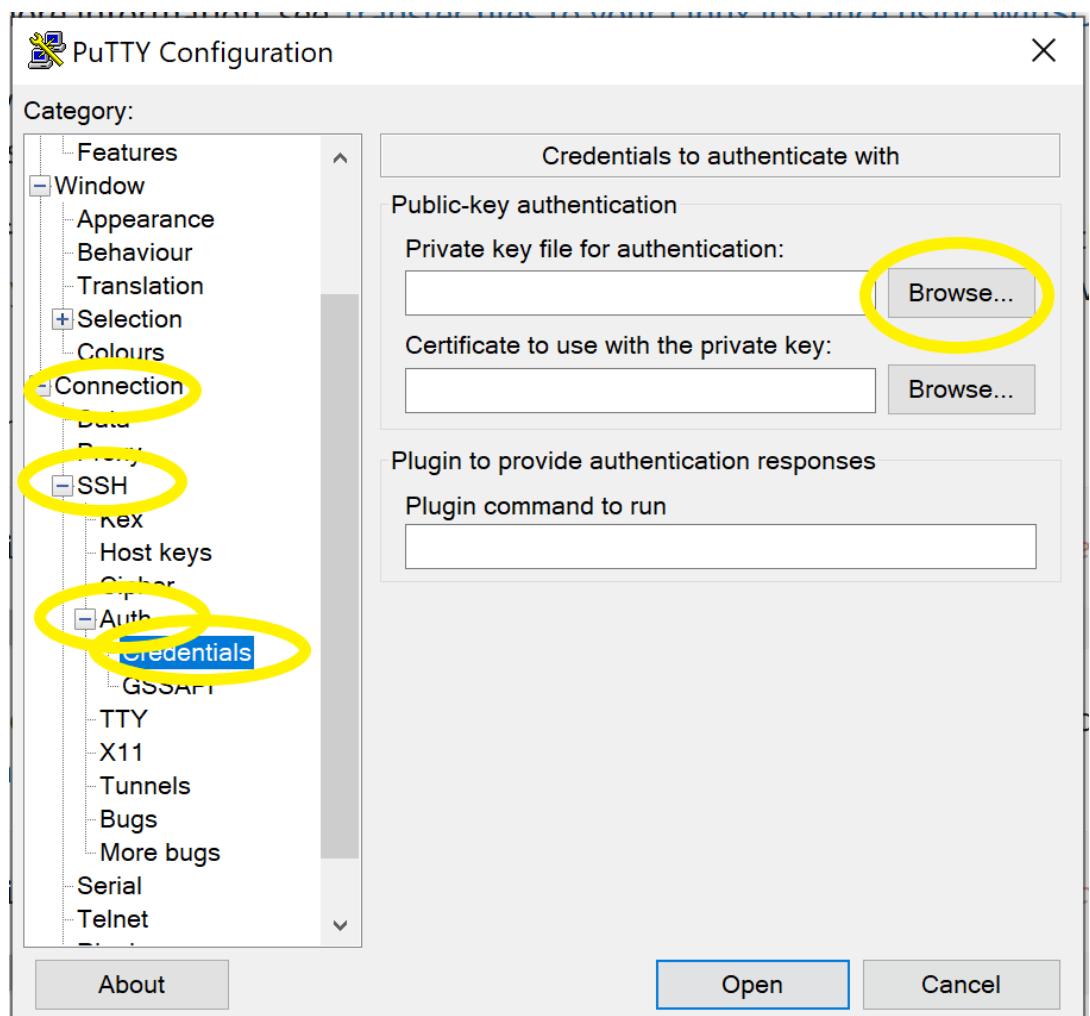
Step 7: Click **Connection** on the left hand side.

Step 8: Expand **SSH**

Step 9: Expand **Auth**

Step 10: Select **Credentials**

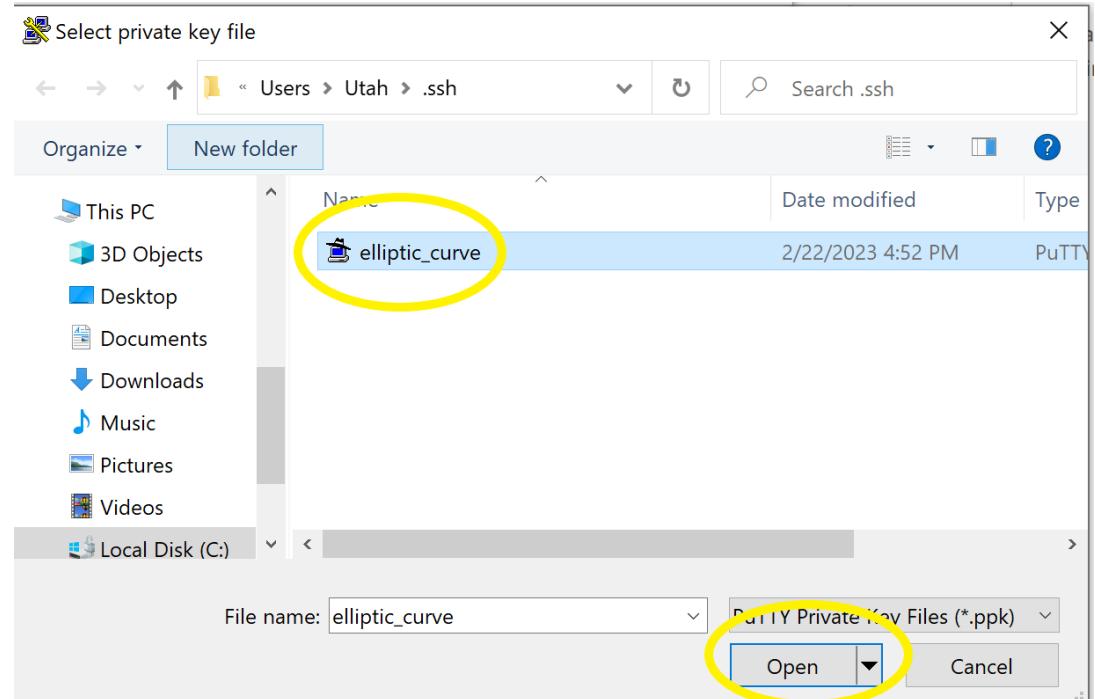
Step 11: Click **Browse** under “Private key file for authentication”



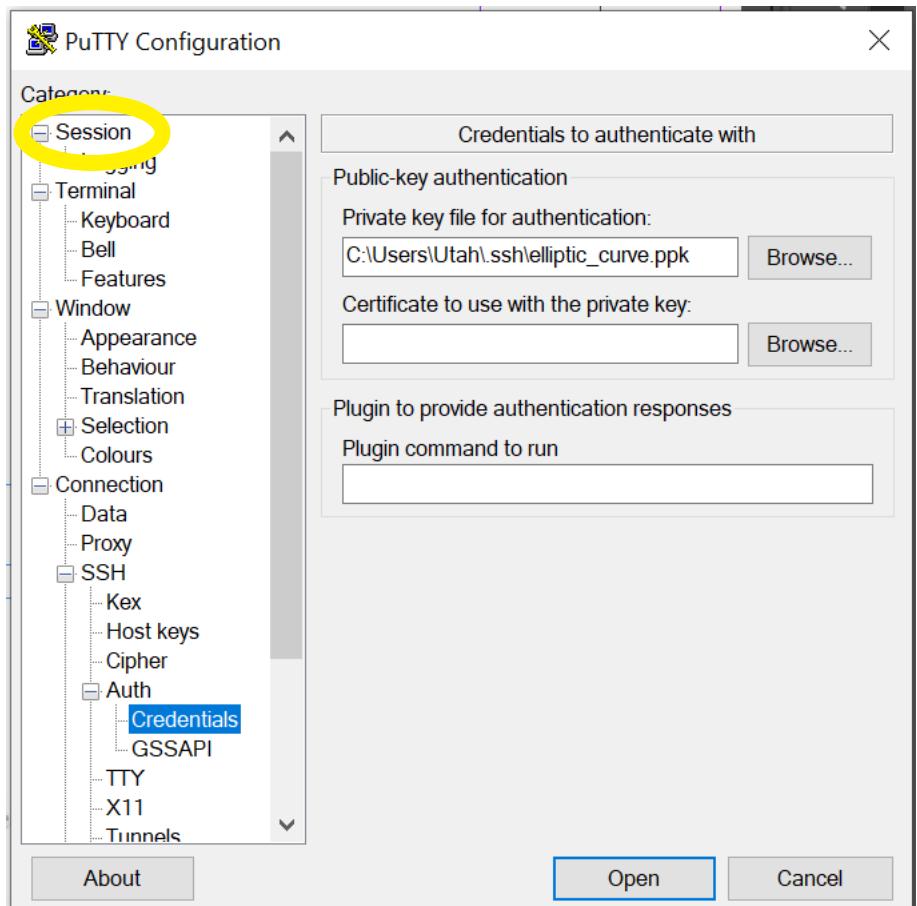
Step 12: Browse to the folder C:\Users\your-username\.ssh.

Step 13: Choose the Key Pair we created earlier.

Step 14: Select “Open”.

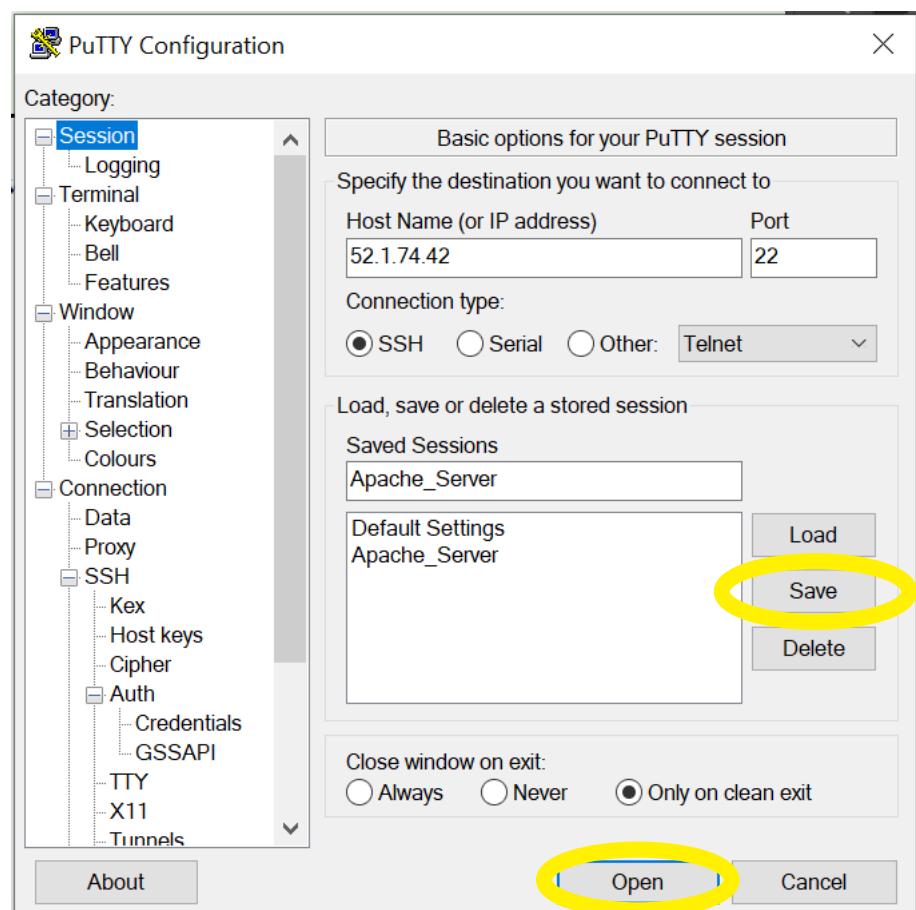


Step 15: Scroll up and Click Session

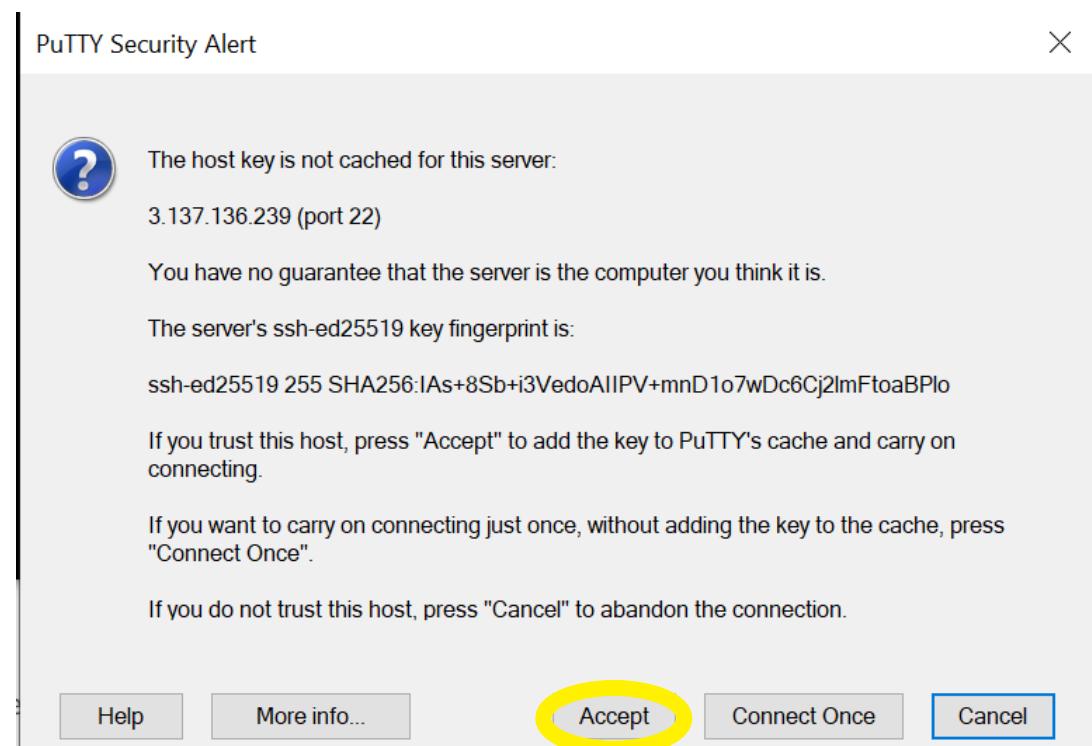


Step 15: Click “Save”

Step 16: Click “Open”



Step 17: Click “Accept”



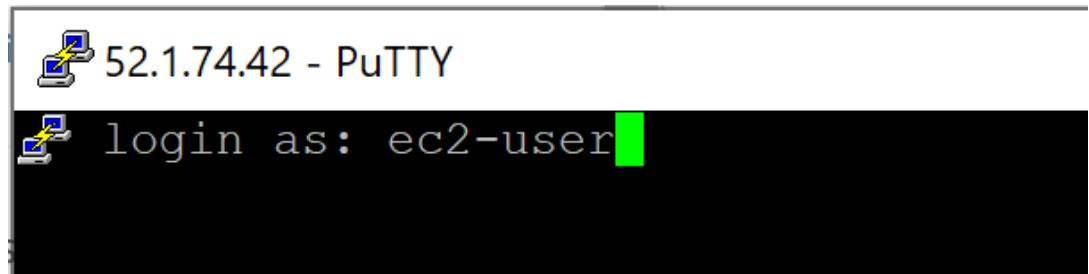
This may require another re-opening of putty after the accept.

Step 17: Enter Login Name: “**ec2-user**” > Hit **”Enter”**

This is the default user for the Amazon Operating System images.

Make sure you it is all lowercase.

Other images such as Ubuntu have different usernames.



Update the OS and Install Apache

This section shows the process for manually patching and installing a webserver. We call it having servers as pets instead of treating them like cattle. Every server is treated with love and tenderness for patching and maintenance.

Step 1: Type "**sudo yum update**" > Hit "**Enter**"

```
ec2-user@ip-172-31-92-37:~  
login as: ec2-user  
Authenticating with public key "elliptic curve"  
Last login: Wed Feb 22 23:25:24 2023 from 104.28.220.4  
  
      _\   _ /     Amazon Linux 2 AMI  
     / \ | |  
  
https://aws.amazon.com/amazon-linux-2/  
No packages needed for security; 5 packages available  
Run "sudo yum update" to apply all updates.  
[ec2-user@ip-172-31-92-37 ~]$ sudo yum update
```

Step 3: Type "**y**" for yes

There may be no new patches to install allowing you to skip this step.

```
ec2-user@ip-172-31-92-37:~  
----> Package tzdata.noarch 0:2022f-1.amzn2.0.1 will be updated  
----> Package tzdata.noarch 0:2022g-1.amzn2.0.2 will be an update  
--> Finished Dependency Resolution  
  
Dependencies Resolved  
  
=====  
 Package      Arch    Version           Repository      Size  
=====  
 Installing:  
   kernel       x86_64  5.10.167-147.601.amzn2  amzn2extra-kernel-5.10  33 M  
 Updating:  
   ec2-hibernate-agent  noarch 1.0.2-4.amzn2          amzn2-core            14 k  
   kernel-tools      x86_64  5.10.167-147.601.amzn2  amzn2extra-kernel-5.10 187 k  
   rsyslog         x86_64  8.24.0-57.amzn2.2.0.2        amzn2-core            619 k  
   tzdata          noarch  2022g-1.amzn2.0.2          amzn2-core            480 k  
  
Transaction Summary  
=====  
Install 1 Package  
Upgrade 4 Packages  
  
Total download size: 34 M  
Is this ok [y/d/N]:
```

Step 4: Type "sudo yum install httpd" > Hit "Enter"

```
[ec2-user@ip-172-31-92-37:~]
[ec2-user login as: ec2-user
[ec2-user Authenticating with public key "elliptic_curve"
Last login: Wed Feb 22 23:43:56 2023 from 104.28.220.4

[ec2-user@ip-172-31-92-37:~]$
[ec2-user@ip-172-31-92-37:~]$ sudo yum install httpd
```

Step 6: Type "y" to continue installation

```
[ec2-user@ip-172-31-88-60:~]$ sudo yum install httpd
Last metadata expiration check: 0:02:33 ago on Fri Apr 14 02:30:12 2023.
Dependencies resolved.
=====
Package          Arch      Version       Repository      Size
=====
Installing:
httpd           x86_64    2.4.56-1.amzn2023   amazonlinux   48 k
Installing dependencies:
apr              x86_64    1.7.2-2.amzn2023.0.2  amazonlinux   129 k
apr-util         x86_64    1.6.3-1.amzn2023.0.1  amazonlinux   98 k
generic-logos-httpd noarch   18.0.0-12.amzn2023.0.3  amazonlinux   19 k
httpd-core       x86_64    2.4.56-1.amzn2023   amazonlinux   1.4 M
httpd-filesystem noarch   2.4.56-1.amzn2023   amazonlinux   15 k
httpd-tools       x86_64    2.4.56-1.amzn2023   amazonlinux   82 k
libbrotli        x86_64    1.0.9-4.amzn2023.0.2  amazonlinux   315 k
mailcap          noarch   2.1.49-3.amzn2023.0.3  amazonlinux   33 k
Installing weak dependencies:
apr-util-openssl x86_64    1.6.3-1.amzn2023.0.1  amazonlinux   17 k
mod_http2        x86_64    2.0.11-2.amzn2023   amazonlinux   150 k
mod_lua          x86_64    2.4.56-1.amzn2023   amazonlinux   62 k
=====
Transaction Summary
=====
Install 12 Packages

total download size: 2.1 M
Installed size: 6.9 M
Is this ok [y/N]: y
```

Step 7: Type "**sudo systemctl enable httpd --now**" > Hit "Enter"

Step 8: Type "**sudo systemctl status httpd**" > Hit "Enter"

Verify that the service is enabled and running with systemctl.

```
[ec2-user@ip-172-31-88-60 ~]$ sudo systemctl enable httpd
Failed to enable unit: Access denied
[ec2-user@ip-172-31-88-60 ~]$ sudo systemctl enable httpd
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/httpd.service.
[ec2-user@ip-172-31-88-60 ~]$ sudo systemctl start httpd
[ec2-user@ip-172-31-88-60 ~]$ sudo systemctl status httpd
● httpd.service - The Apache HTTP Server
    Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; preset: disabled)
      Active: active (running) since Fri 2023-04-14 02:41:11 UTC; 4s ago
        Docs: man:httpd.service(8)
       Main PID: 25557 (httpd)
          Status: "Started, listening on: port 80"
             Tasks: 177 (limit: 1112)
            Memory: 12.8M
              CPU: 69ms
            CGroup: /system.slice/httpd.service
                    ├─25557 /usr/sbin/httpd -DFOREGROUND
                    ├─25558 /usr/sbin/httpd -DFOREGROUND
                    ├─25559 /usr/sbin/httpd -DFOREGROUND
                    ├─25560 /usr/sbin/httpd -DFOREGROUND
                    └─25561 /usr/sbin/httpd -DFOREGROUND

Apr 14 02:41:11 ip-172-31-88-60.ec2.internal systemd[1]: Starting httpd.service - The Apache HTTP Server...
Apr 14 02:41:11 ip-172-31-88-60.ec2.internal systemd[1]: Started httpd.service - The Apache HTTP Server...
Apr 14 02:41:11 ip-172-31-88-60.ec2.internal httpd[25557]: Server configured, listening on: port 80
```

Step 9: Hit "**Control C**" to exit this screen

Fix our Security Group to allow HTTP/s

After enabling Apache on the server we need to now go into the security group we made and permit HTTP and HTTPS traffic.

Step 1: Search **"Security groups"**

Step 2: Click **"Security groups"**

Search results for 'security groups'

Services

- EC2 ★ Virtual Servers in the Cloud
- IAM ★ Manage access to AWS resources
- AWS Firewall Manager ★ Central management of firewall rules
- Security Lake ★ Automatically centralize all your security data with a few clicks

Features

- Security groups ★ EC2 feature

Step 3: Find the **Security Group** we wrote down earlier.

Step 4: Click its **"Security group ID"**

Name	Security group ID	Security group name	VPC ID	Description	Owner
-	sg-085041a89247229b5	launch-wizard-1	vpc-20d2235d	launch-wizard-1 create...	941897908476

Step 5: Click **"Edit Inbound Rules"**

Details

Security group name: launch-wizard-2
Owner: 941897908476
Security group ID: sg-0ad870cb70894c2ec
Description: launch-wizard-2 created 2025-04-14T02:28:14.478Z
VPC ID: vpc-20d2235d

Inbound rules (1/1)

Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description
-	sg-03c1a1311c9bedd6	IPv4	SSH	TCP	22	0.0.0.0/0	-

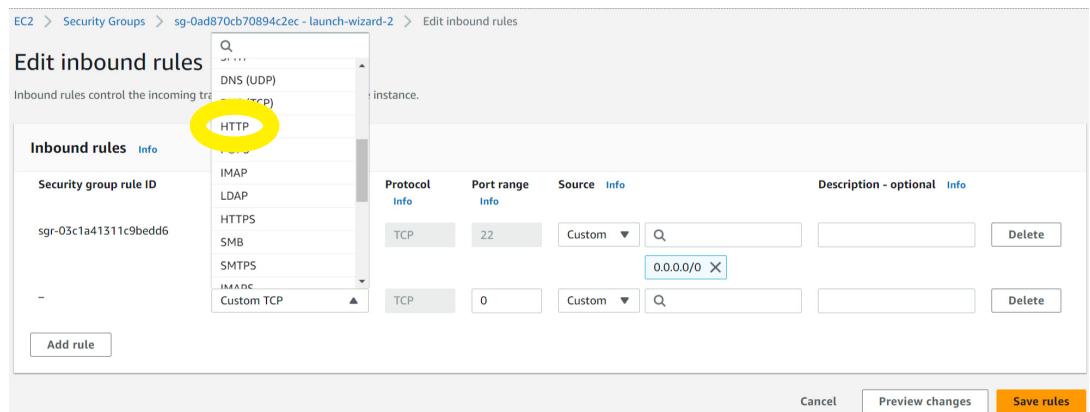
Step 6: Click “Add Rule”

The screenshot shows the 'Edit inbound rules' section of the AWS EC2 Security Groups interface. At the top, there's a breadcrumb navigation: EC2 > Security Groups > sg-0ad870cb70894c2ec - launch-wizard-2 > Edit inbound rules. Below the navigation, the title 'Edit inbound rules' is displayed with a 'Info' link. A descriptive text states: 'Inbound rules control the incoming traffic that's allowed to reach the instance.' The main area is titled 'Inbound rules' with an 'Info' link. It contains a table with columns: 'Security group rule ID', 'Type', 'Protocol', and 'Port range'. One row is shown: 'sgr-03c1a41311c9bedd6' (Type: SSH, Protocol: TCP, Port range: 22). At the bottom left, a prominent 'Add rule' button is highlighted with a yellow oval.

Step 7: Click “Type Drop Down”

This screenshot continues from Step 6. It shows the same 'Edit inbound rules' interface after a new rule has been added. The table now includes a second row: '- (Type: Custom TCP, Protocol: TCP, Port range: 0, Source: Custom)'. The 'Add rule' button at the bottom left is still present. The 'Type' dropdown for the new rule is highlighted with a yellow oval.

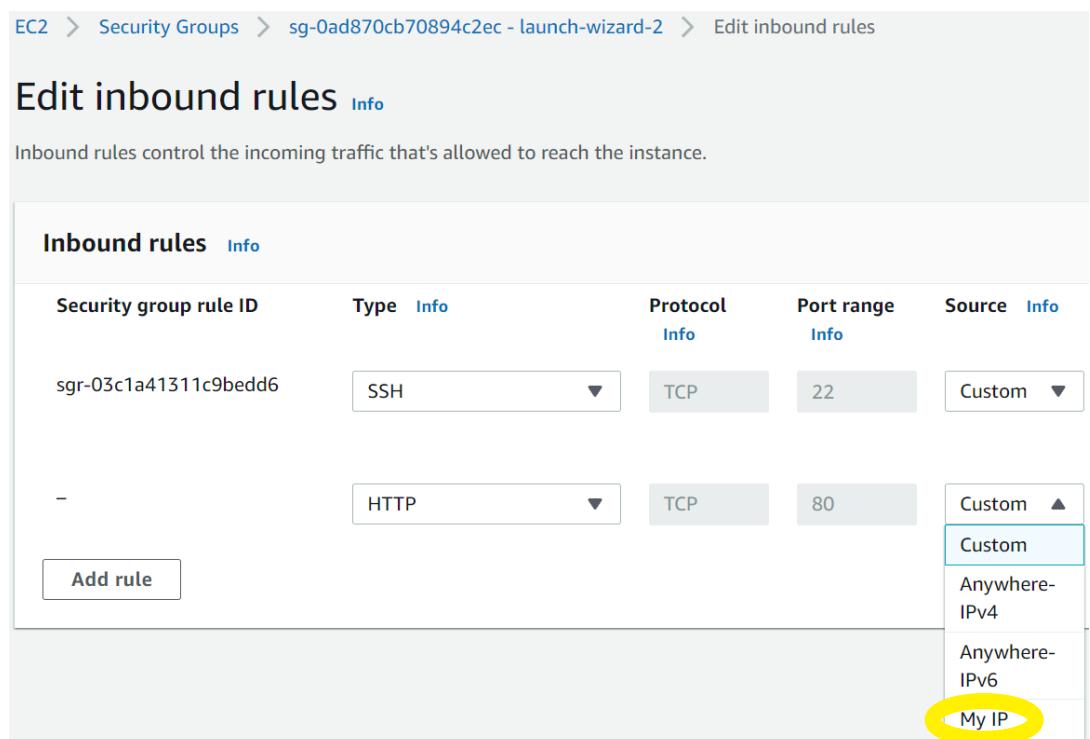
Step 8: Click “HTTP”



The screenshot shows the 'Edit inbound rules' interface for a security group. On the left, there's a sidebar with a dropdown menu showing options like DNS (UDP), HTTP, IMAP, LDAP, HTTPS, SMB, and SMTPS. The 'HTTP' option is highlighted with a yellow circle. The main area lists two existing rules: one for port 22 (TCP) and another for port 0 (TCP). At the bottom right, there are 'Cancel', 'Preview changes', and a prominent orange 'Save rules' button.

Step 9: Click “Custom”

Step 10: Click “My IP”



This screenshot shows the same 'Edit inbound rules' interface. It displays two rules: one for SSH (TCP port 22) with a 'Custom' source, and another for HTTP (TCP port 80) with a 'Custom' source. The 'Custom' source dropdown for the second rule is expanded, showing options like 'Anywhere-IPv4', 'Anywhere-IPv6', and 'My IP'. The 'My IP' option is highlighted with a yellow circle. The 'Add rule' button is visible at the bottom left.

Step 11: Click the “Type Drop Down” and select “HTTPS”

Step 12: Click “Custom”

Step 13: Click “My IP”

The screenshot shows the 'Edit inbound rules' page for a specific security group. The 'Inbound rules' table lists three rules:

Security group rule ID	Type	Protocol	Port range	Source
sgr-0c39ff5f75aa0ab3c	SSH	TCP	22	My IP
-	HTTP	TCP	80	My IP
-	HTTPS	TCP	443	My IP

Below the table is an 'Add rule' button.

Step 8: Click “Save Rules”

The screenshot shows the 'Edit inbound rules' page with three rules listed:

Security group rule ID	Type	Protocol	Port range	Source
sgr-0c39ff5f75aa0ab3c	SSH	TCP	22	My IP
-	HTTP	TCP	80	My IP
-	HTTPS	TCP	443	My IP

At the bottom right of the page, there are 'Cancel', 'Preview changes', and 'Save rules' buttons. A yellow circle highlights the 'Save rules' button.

Install and Modify the Server Software Firewall

We need to install the software firewall on the server as well as open holes for ssh, HTTP, and HTTPS.

Relaunch the Putty Application

Step 1: Type “`sudo dnf --assumeyes install firewalld`” > Hit “Enter”

```
[ec2-user@ip-172-31-88-60 ~]$ sudo dnf --assumeyes install firewalld
Last metadata expiration check: 0:46:58 ago on Fri Apr 14 02:30:12 2023.
Dependencies resolved.
=====
| Package           | Architecture | Version   | Repository | Size |
|=====|
| Installing:      |             |           |            |        |
| firewalld         | noarch      | 1.2.3-1.amzn2023 | amazonlinux | 452 k |
| Installing dependencies: |             |           |            |        |
| firewalld-filesystem | noarch      | 1.2.3-1.amzn2023 | amazonlinux | 11 k  |
| gobject-introspection | x86_64     | 1.73.0-2.amzn2023.0.2 | amazonlinux | 255 k |
| ipset              | x86_64     | 7.11-1.amzn2023.0.3 | amazonlinux | 40 k  |
| ipset-libs         | x86_64     | 7.11-1.amzn2023.0.3 | amazonlinux | 67 k  |
| iptables-libs       | x86_64     | 1.8.8-3.amzn2023.0.2 | amazonlinux | 401 k |
| iptables-nft        | x86_64     | 1.8.8-3.amzn2023.0.2 | amazonlinux | 183 k |
| libnetfilter_conntrack | x86_64     | 1.0.8-2.amzn2023.0.2 | amazonlinux | 58 k  |
| libnftnl            | x86_64     | 1.0.1-19.amzn2023.0.2 | amazonlinux | 30 k  |
| libnftnl            | x86_64     | 1.2.2-2.amzn2023.0.2 | amazonlinux | 84 k  |
| nftables             | x86_64     | 1:1.0.4-3.amzn2023.0.2 | amazonlinux | 400 k |
| python3-firewall    | noarch      | 1.2.3-1.amzn2023 | amazonlinux | 357 k |
| python3-gobject-base | x86_64     | 3.42.2-2.amzn2023.0.2 | amazonlinux | 179 k |
| python3-gobject-base-noarch | noarch      | 3.42.2-2.amzn2023.0.2 | amazonlinux | 154 k |
| python3-nftables    | x86_64     | 1:1.0.4-3.amzn2023.0.2 | amazonlinux | 18 k  |
| Installing weak dependencies: |             |           |            |        |
| libcap-ng-python3    | x86_64     | 0.8.2-4.amzn2023.0.2 | amazonlinux | 30 k  |
| Transaction Summary |             |           |            |        |
|=====|
| Install 16 Packages |             |           |            |        |
| Total download size: 2.7 M |             |           |            |        |
| Installed size: 11 M |             |           |            |        |
| Downloading Packages: |             |           |            |        |
|=====|
```

Step 2: Type “`sudo systemctl enable firewalld --now`” > Hit “Enter”

Step 3: Type “`sudo firewall-cmd --zone=public --permanent --add-service=ssh`” > Hit “Enter”

Step 4: Type “`sudo firewall-cmd --zone=public --permanent --add-service=http`” > Hit “Enter”

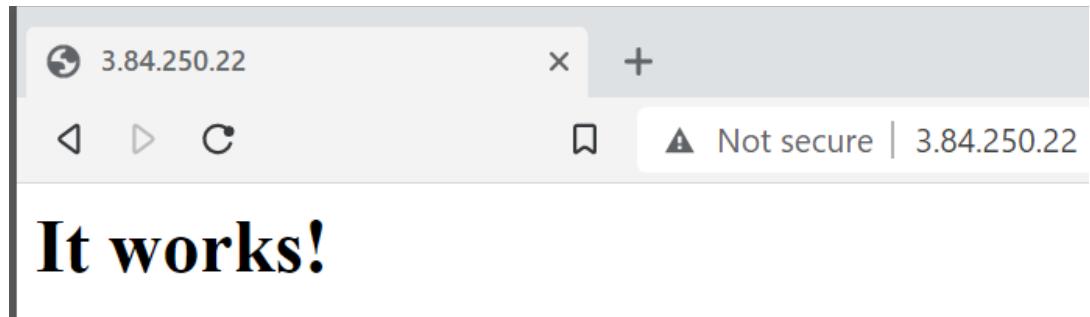
Step 5: Type “`sudo firewall-cmd --zone=public --permanent --add-service=https`” > Hit “Enter”

Step 6: Type “`sudo systemctl reload firewalld`” > Hit “Enter”

Step 7: Type “`sudo systemctl status firewalld`” > Hit “Enter”

```
[ec2-user@ip-172-31-88-60 ~]$ sudo systemctl start firewalld
[ec2-user@ip-172-31-88-60 ~]$ sudo firewall-cmd --zone=public --permanent --add-service=ssh
Warning: ALREADY_ENABLED: ssh
success
[ec2-user@ip-172-31-88-60 ~]$ sudo firewall-cmd --zone=public --permanent --add-service=http
success
[ec2-user@ip-172-31-88-60 ~]$ sudo firewall-cmd --zone=public --permanent --add-service=https
success
[ec2-user@ip-172-31-88-60 ~]$ sudo systemctl enable firewalld
[ec2-user@ip-172-31-88-60 ~]$ sudo systemctl reload firewalld
[ec2-user@ip-172-31-88-60 ~]$
```

Step 8: Open your web browser and enter the server IP into the address bar.

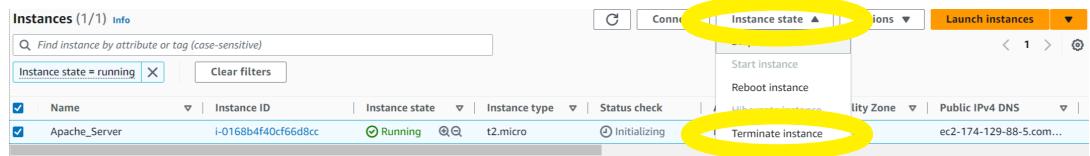


Delete the Server (Instance)

Deleting all the changes so we do not incur costs. You can also shutdown the instance leaving only storage costs rather than running instance costs.

Step 1: Click “**Instant State**”

Step 2: Click “**Terminate Instance**”



Save the Key-Pair

We will use the Key Pair for Part 2: Infrastructure as Code with Terraform.

Delete the Security Group

The security group can only be deleted after the Instance has been deleted.

Step 1: Click “**Security**”

Step 2: Click the Security Group in Blue

The screenshot shows the AWS EC2 Instances details page for an instance with ID i-0168b4f40cf66d8cc. The page is titled "Instance summary for i-0168b4f40cf66d8cc (". The "Security" tab is highlighted with a yellow oval. Below it, the "Security groups" section shows a blue link "sg-01713e32461f14b58 (launch-wizard-3)".

EC2 > Instances > i-0168b4f40cf66d8cc

Instance summary for i-0168b4f40cf66d8cc (Updated less than a minute ago

Instance ID
i-0168b4f40cf66d8cc (Apache_Server)

IPv6 address
-

Hostname type
IP name: ip-172-31-83-82.ec2.internal

Answer private resource DNS name
IPv4 (A)

Auto-assigned IP address
174.129.88.5 [Public IP]

IAM Role
-

IMDSv2
Required

Details **Security** Networking Storage

▼ Security details

IAM Role
-

Security groups
sg-01713e32461f14b58 (launch-wizard-3)

Step 3: Click “Action”

Step 4: Click “Delete security groups”

The screenshot shows the AWS EC2 Security Groups interface. In the top navigation bar, 'EC2' and 'Security Groups' are selected, with the specific group 'sg-01713e32461f14b58 - launch-wizard-3' highlighted. The main title 'sg-01713e32461f14b58 - launch-wizard-3' is displayed above a 'Details' table. The 'Actions' menu, located in the top right corner, is open and lists several options: 'Edit inbound rules', 'Edit outbound rules', 'Manage tags', 'Copy to new security group', and 'Delete security groups'. The 'Delete security groups' option is the one being targeted.

Details		
Security group name launch-wizard-3	Security group ID sg-01713e32461f14b58	Description launch-wizard-3 created 2023-04-17T10:41:10Z

- Actions ▾
- Edit inbound rules
- Edit outbound rules
- Manage tags
- Copy to new security group
- Delete security groups**

Intro to Cloud 101

Part 2 Infrastructure as Code with Terraform

Target Audience: Non Programmers with little to no experience with cloud automation or general cloud use. No previous experience is required.

Disclaimer: This is an ultra simple guide. Breaking the one “terraform.tf” file up into separate parts is normal practice. It is also normal to use modules. We will not be doing either of those for simplicity sake.

Presenter: David Thurm

Contact:

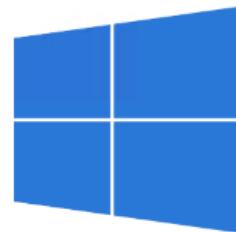
LinkedIn: [davidthurm](#)

Twitter: [@davidthurm](#)

VScode Install

Step 1: Download Code

Step 2: Click “**User Installer x64**”



User Installer	x64	x86	Arm64
System Installer	x64	x86	Arm64
.zip	x64	x86	Arm64
CLI	x64 x86 Arm64		

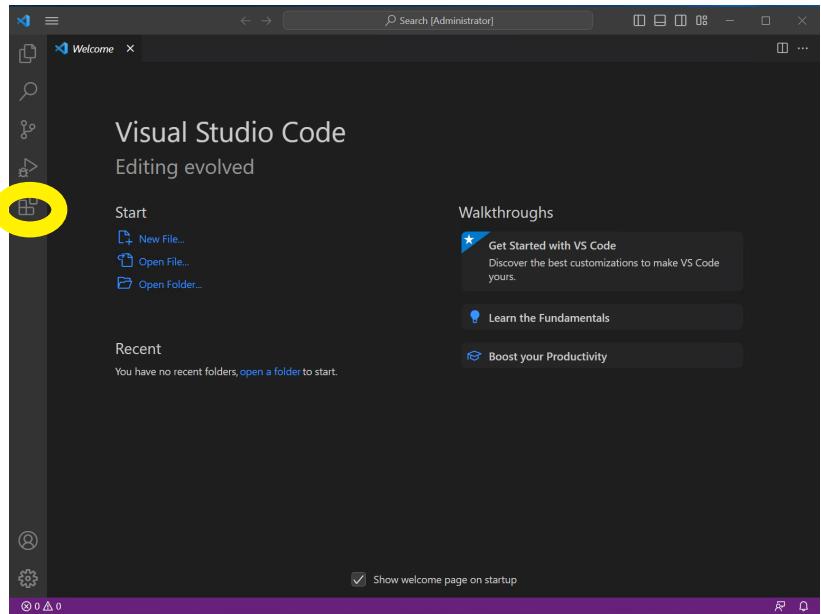
Step 3: Open the Downloaded File



Step 4: Open the Downloaded File

Step 5: Continue as normal as with any software installation.

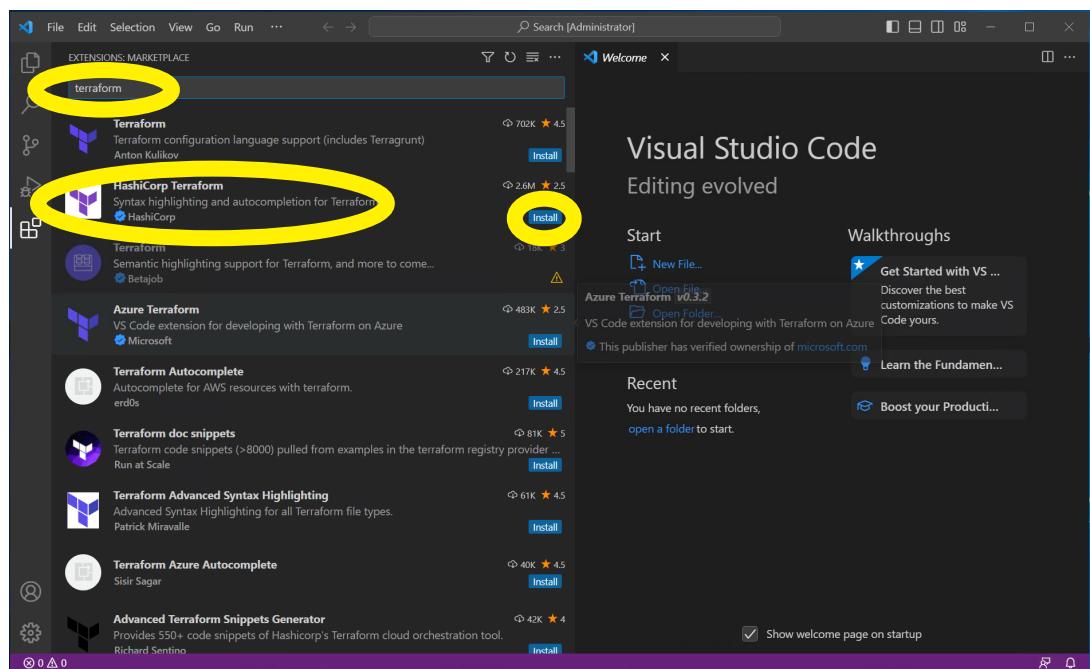
Step 6: Click “**Extensions**”



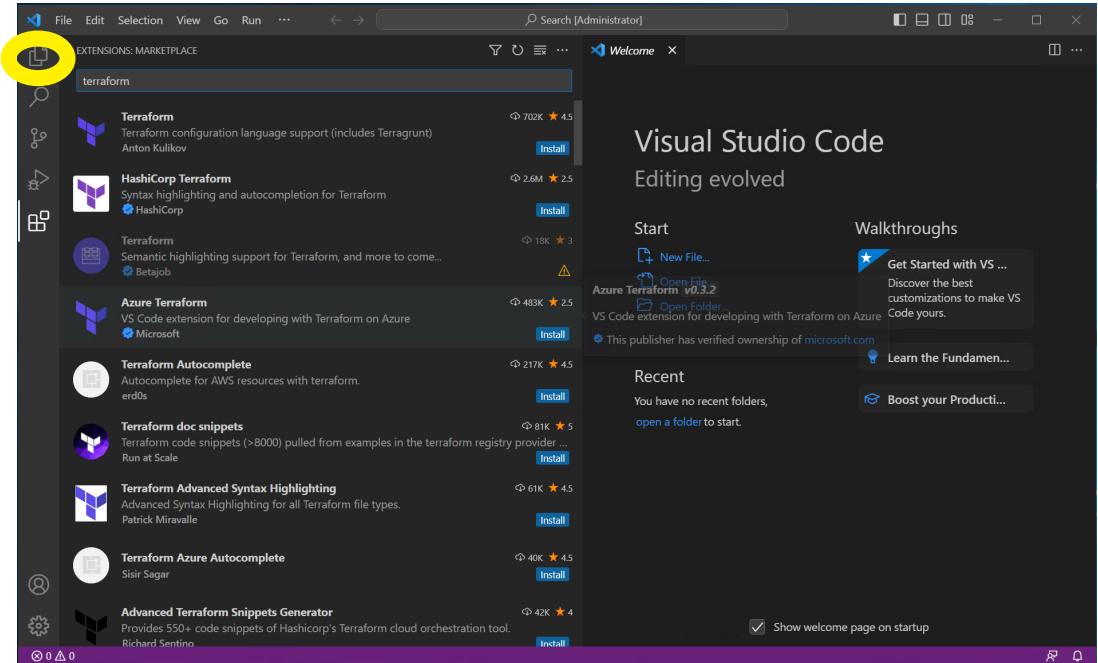
Step 7: Search for “**Terraform**”

Step 8: Verify the vendor is HashiCorp

Step 9: Click “**Install**”

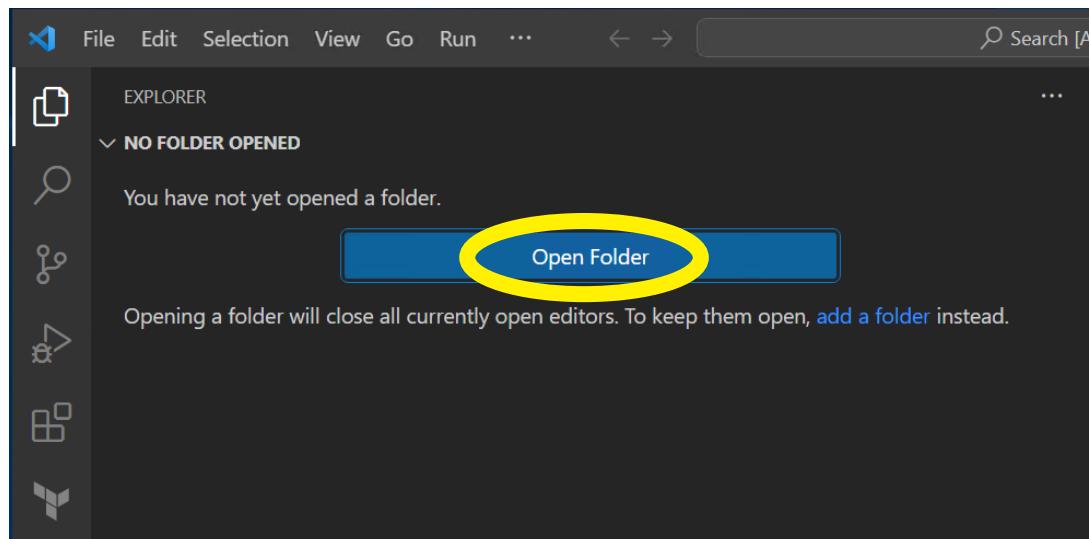


Step 10: Click on “Explorer”



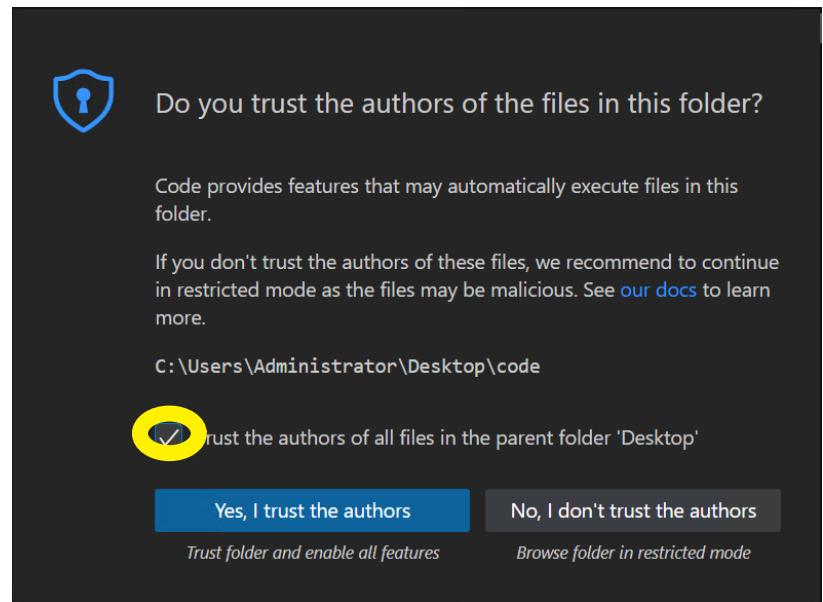
Step 11: Click on “Open Folder”

Step 12: Pick a folder of your choice.

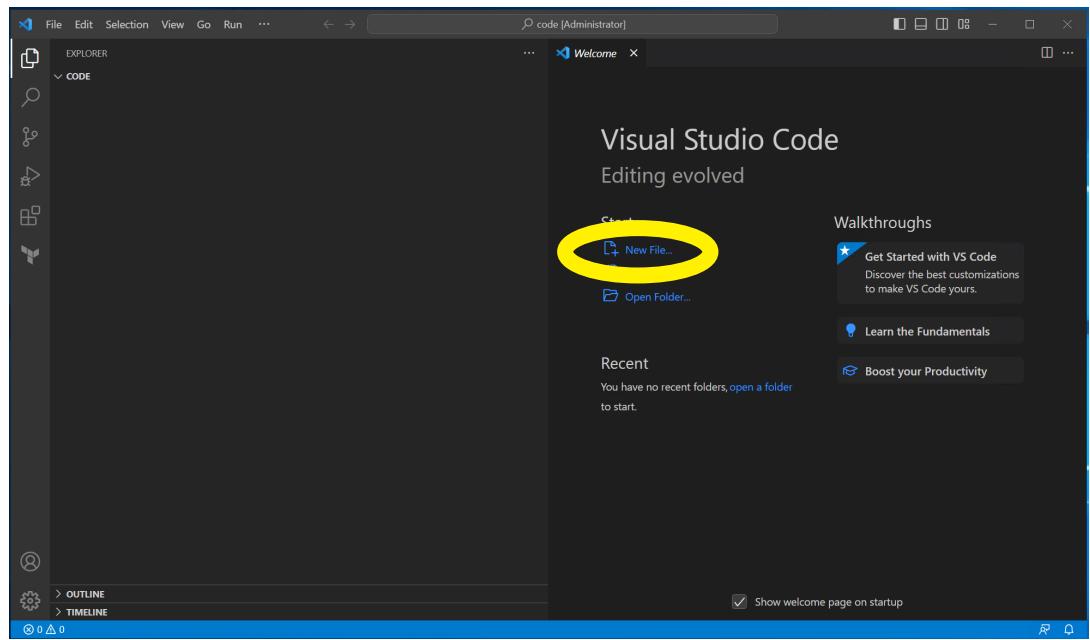


Step 13: Checkmark on “**Trust the authors**”

Step 12: Click “**Yes, Trust the authors**”

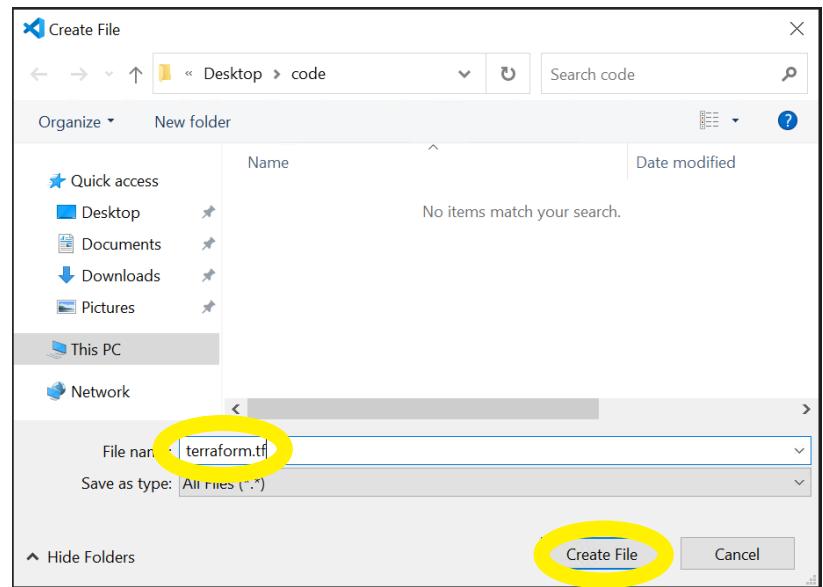


Step 14: Click “New File”

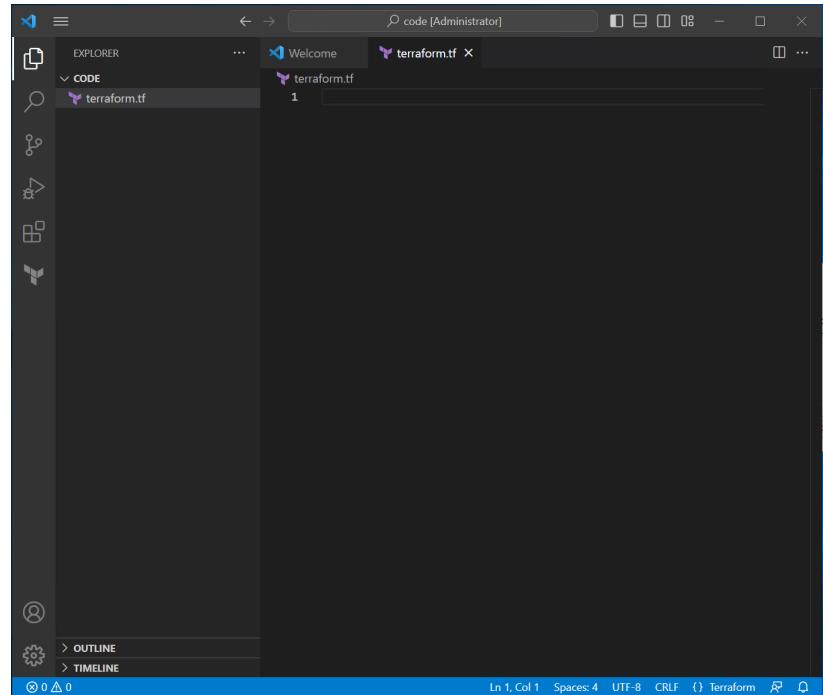


Step 15: Click “New File”

Step 16: Name the File “**terraform.tf**”



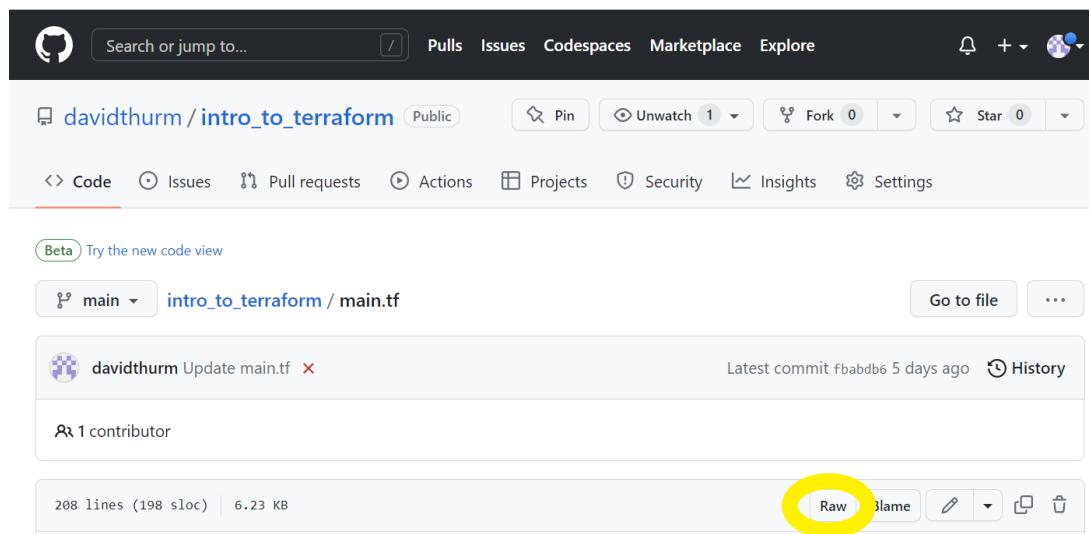
Step 17: We are ready to begin writing Terraform.



Github

Step 1: Browse to https://github.com/davidthurm/intro_to_terraform/blob/main/main.tf

Step 2: Click “RAW”



Intentional Bugs are left in this code allowing for changes to be made later on.

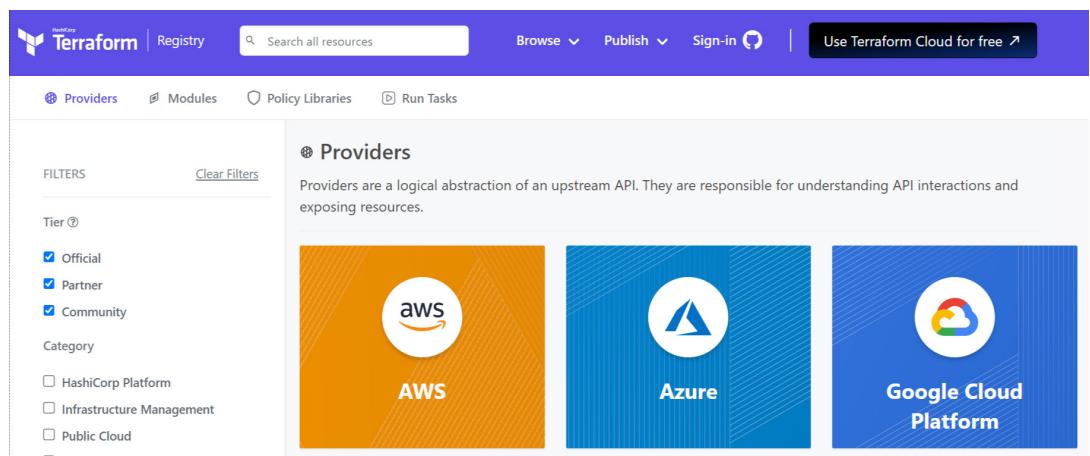
Raw is the preferred choice, often times if we do not choose RAW the code will not copy/paste or execute properly when doing CI/CD pipelines. A specific example is using [AWS Systems Manager Run Command](#) to launch Ansible code on your servers.

Side-by-Side VScode and Github Repository is recommended for copying code section by section as it is modified during the training.

Pulling down the Terraform AWS Provider Configuration

Step 1: Browse to <https://registry.terraform.io/browse/providers>

Step 2: Click “AWS”



Step 3: Browse to <https://registry.terraform.io/browse/providers>

Step 4: Click “**USE PROVIDER**”

Step 5: Copy the code highlighted in the Yellow Box to your blank terraform.tf file.

The screenshot shows the HashiCorp Terraform Registry interface. In the top right corner, there is a yellow circle highlighting the "USE PROVIDER" button. On the left, the AWS provider page is displayed, showing its official status and version 4.63.0. On the right, a box highlights the Terraform configuration code for using the AWS provider:

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "4.63.0"
    }
  }
}

provider "aws" {
  # Configuration options
}
```

Setting the Mandatory Terraform Sections

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "4.62.0"
    }
  }
}

provider "aws" {
  region = "us-east-1" // Change to your desired region
}
```

A yellow box labeled "Required" surrounds the "aws" provider block. A blue box labeled "Optional" surrounds the "region" configuration line.

Terraform Providers are plugins that are required to interact with cloud Platforms.

Provider Regions directly correlate with AWS datacenter regions. Amazon releases newer features into the US East region earlier and is one of the original regions. Not all regions have the same AWS applications and features in them.

VPC Configuration -- Optional (Sometimes)

```
#####
## Create the VPC ##
#####
```

← Remark your code

Required Syntax

Name we choose

```
resource "aws_vpc" "example" {
  cidr_block = "10.0.0.0/16" ← CIDR of every IP we want available
  tags = {
    Name = "dev-vpc" ← Label your VPC
  }
}
```

Remark your code at the top of each section for the next person that comes along.

AWS VPCs are virtual cloud environments. It is comprised of virtual network routers/switches, storage (s3, EBS), web and network firewalls, virtual servers. Imagine your own private workspace.

resource "aws_vpc" identifies that we want to create a VPC.

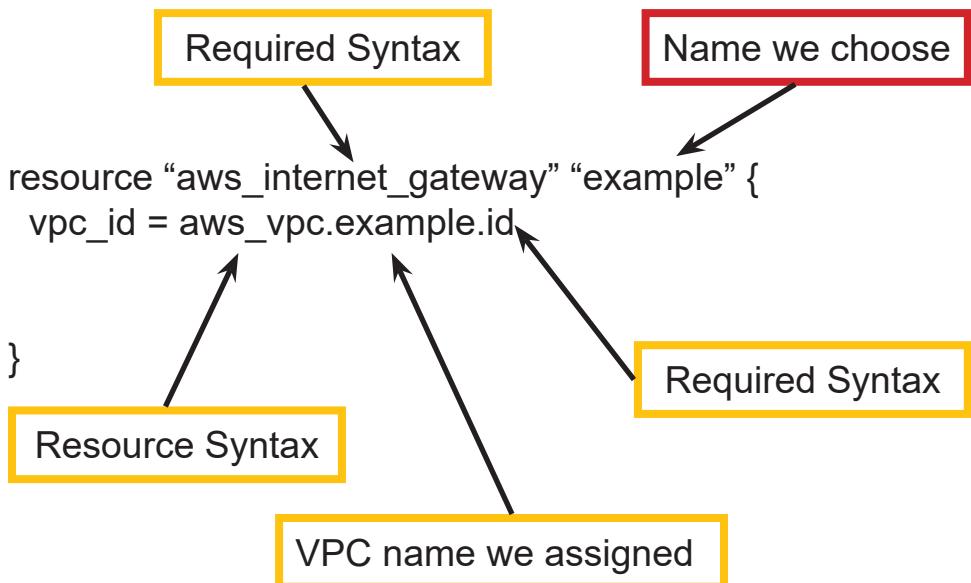
example is a name chosen to describe the VPC. Pick something that is easy to understand and will be repeated throughout the code.

cidr_block is a mandatory setting stating available private IP addresses that are going to be inside the VPC.

tags are used to label pieces of the cloud configuration as well as run additional automated tasks against items with a specific tag. They are highly important and it is best to standardize tagging on day one of your cloud migration.

Internet Gateway -- Optional (Sometimes)

```
#####
## Create the Internet Gateway ##
#####
```



[resource aws_internet_gateway](#) identifies that we will be creating a new gateway the the name of resource name of “example”. Choose your own resource name if you would like. Without this setting our VPC is completely isolated from the Internet and is inaccesable from both inbound and outbound.

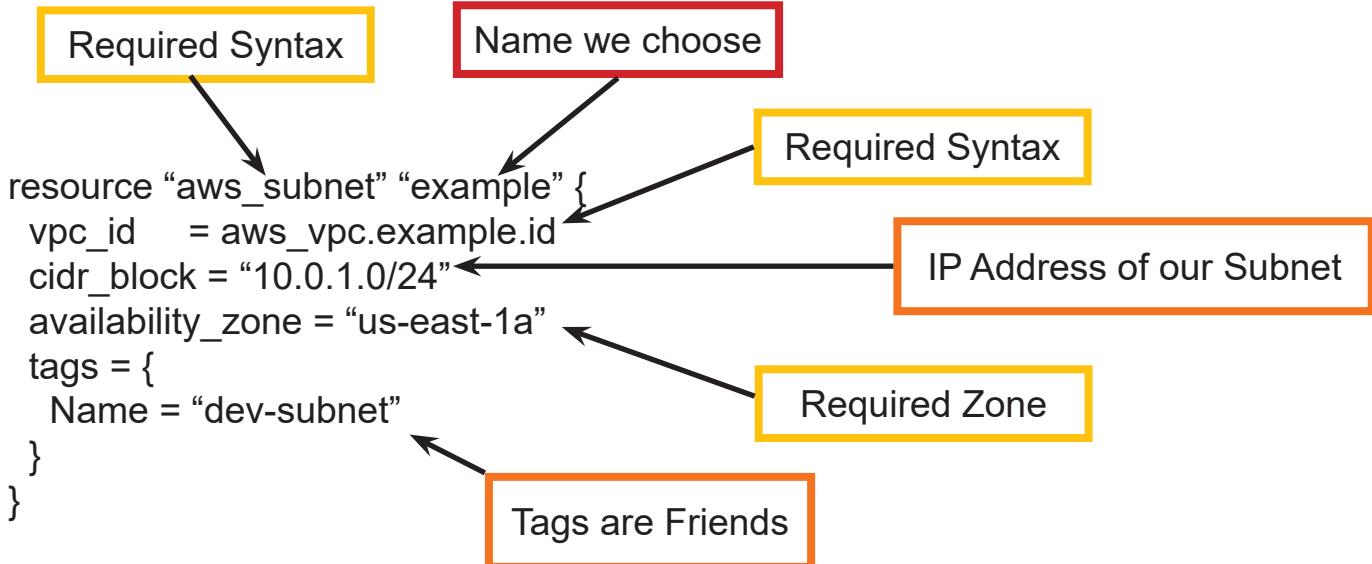
[vpc_id](#) refers to the vpc we created above. This is determined by the resource identifier of “aws_vpc” combined with the name we gave the VPC resource and the syntax “id” at the end. The “id” syntax is derived form the vpc_id variable that is automatically generated when we create the VPC.

[tags](#) can be added as you see fit.

```
tags = {
  Name      = "igw-main"
  environment = "dev"
  team      = "platform security"
  owner     = "john@doe.com"
}
```

Subnet Configuration

```
#####
## Create the Subnet ##
#####
```



resource aws_subnet creates the subnet we will use to house our EC2 instance.

vpc_id will match the previous configuration for the Internet Gateway.

cidr_block this space needs to sit inside the CIDR_block we attached to the VPC we created.

availability_zone needs to match the zone you wish to deploy your EC2 instance in.

tags as your standards dictate. The setting of “Name” is case sensitive and will show up in the AWS console gui as the name of the resource. This may have changed though.

Global Route Table Configuration

```
#####
## Create the Route Table ##
#####
```

```
resource "aws_route_table" "example" {
  vpc_id = aws_vpc.example.id
  tags = {
    Name = "example-route-table"
  }
}
```

`resource aws_route_table` creates the default routing table for your VPC. The EC2 instance subnet will be added to this table as well as the default route outbound to the Internet.

`vpc_id` refers to the vpc we created above.

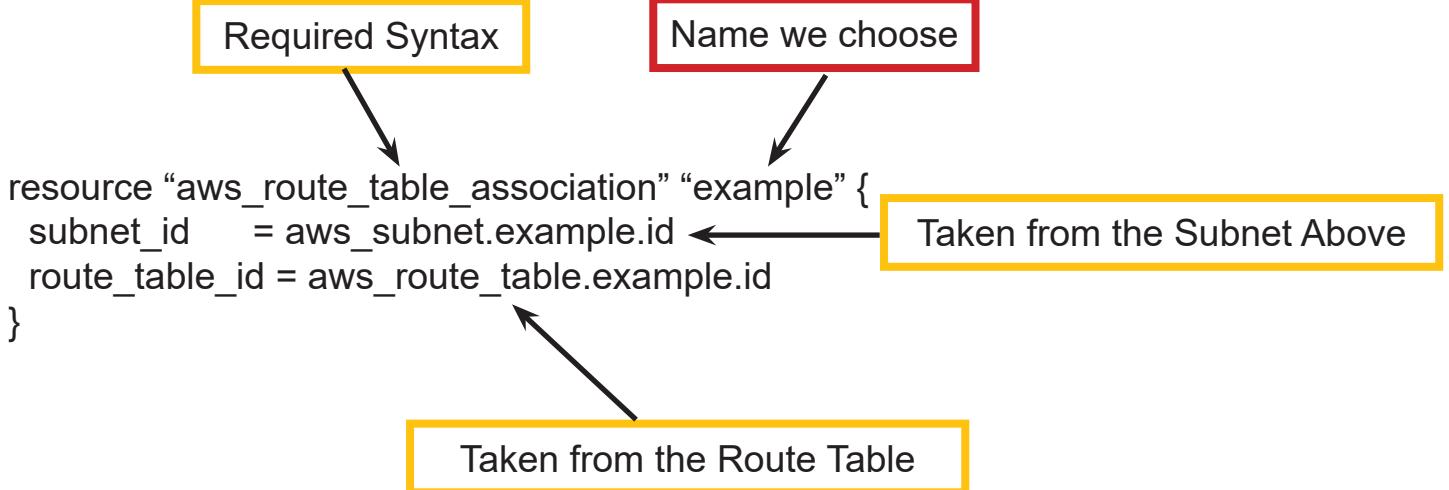
Default Route Configuration

```
#####
## Create the Default Route ##
#####
```

```
resource "aws_route" "default_route" {
  route_table_id      = aws_route_table.example.id
  destination_cidr_block = "0.0.0.0/0"
  gateway_id          = aws_internet_gateway.example.id
}
```

Route Association Configuration

```
#####
## Create the Route Association ##
#####
```



[resource aws_route_table_association](#) places the EC2 subnet into the global routing table along with the default Internet Gateway.

Security Group Configuration

```
#####
## Create the Security Group to allow SSH ##
## Modify the cider_blocks to your home IP ##
#####

Required Syntax          Name we choose
resource "aws_security_group" "example" {
    name_prefix = "dev-ssh-access" <-- Name the Security Group

    ingress {
        from_port = 22
        to_port   = 22
        protocol  = "tcp"
        cidr_blocks = ["66.0.0.97/32"] # Change this to your home ip
    }
    ingress {
        from_port = 80
        to_port   = 80
        protocol  = "tcp"
        cidr_blocks = ["66.0.0.97/32"] # Change this to your home ip
    }
    ingress {
        from_port = 443
        to_port   = 443 <-- Standard Destination Port
        protocol  = "tcp"
        cidr_blocks = ["66.0.0.97/32"] # Change this to your home ip
    }
    egress {
        from_port  = 0 <-- Allow Any Port Outbound
        to_port    = 0
        protocol   = "-1"
        cidr_blocks = ["0.0.0.0/0"] ## Allow all outbound traffic
    }
    tags = {
        Name = "dev-security-group" <-- Standard Wildcard
    }
    vpc_id = aws_vpc.example.id <-- VPC Variable
}
```

[aws_security_group](#) sets the configuration for the Access Control List (ACL) we want to have in front of our EC2 Instance. This is not a stateful firewall and should be used in addition to a commercial firewall solution in production. It protects traffic inbound and outbound of the Layer 3 subnet. It does not protect traffic between two EC2 instances on the same subnet.

Gather AMI (OS Image) Details

```
#####
## Search for newest Amazon OS Image ##
#####
```

```
Required Syntax      Name we choose
↓                  ↓
data "aws_ami" "latest_amazon_linux_2" {
  most_recent = true
  owners      = ["137112412989"] ← Amazon's Owner ID
  filter {
    name = "name"
    values = ["al2023-ami-2023.*x86_64"] ← Unique Info Within the AMI Name
  }
}
```

[data source](#) is used to perform a search of an existing AWS data source. In this case the data source is `aws_ami`. It will then export the search results under the name “example”.

`most_recent` will identify what result has the newest version of the image.

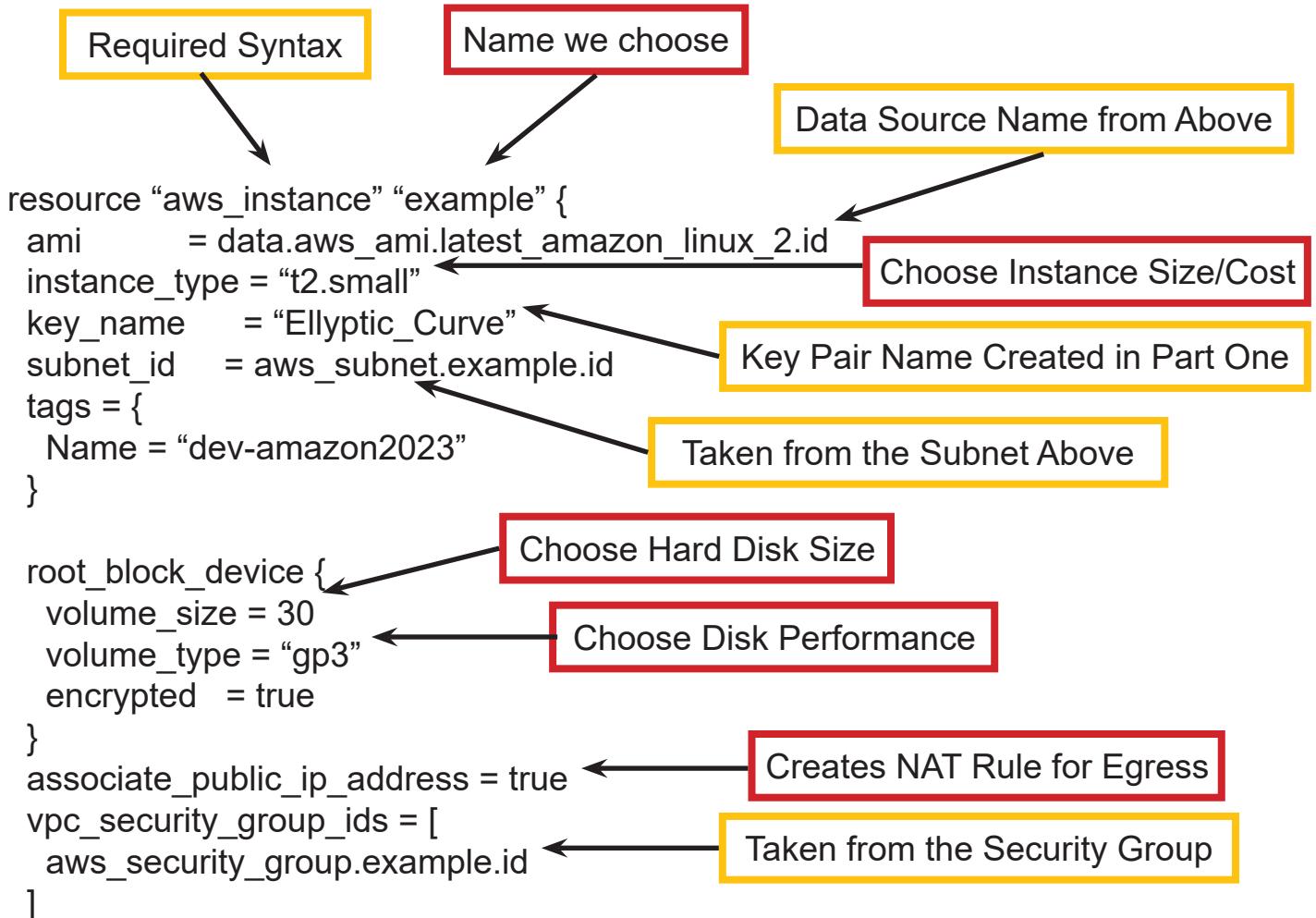
`owners` identifier is used to spell out that only a specific vendor image is required.

`Filter` values is created by doing a search via the “aws cli”. It will require several attempts but a starting point is here:

cli command: “`aws ec2 describe-images --image-id ami-06e46074ae430fba6`”

EC2 Instance Configuration

```
#####
## Create the actual Ec2 Instance ##
#####
```



resource aws_instance initiates the configuration details to be applied to the server. The syntax that follows in this case “example” can be declared as something more useful such as web or fileserver.

ami defines the ami (OS image) we want to deploy. In this case we are using the resource data that we discovered earlier to supply the newest version of Amazon2023 OS ami. This can be substituted for any AMI of your choice by looked up its ID via the AWS web GUI or AWS CLI commands. An example would be a Redhat 9 image “ami-0c41531b8d18cc72b”.

instance_type determines the processing power, network throughput, and amount of memory assigned to your server. This determines **COST**. There are several “Free Tier” instance types available. The first part of the instance_type determines the function of the server. C series servers are for compute intensive workloads while M and T series are general purpose. For most purposes M and T series are preferred giving a balance of compute, memory, and networking. Running servers for limited periods of time does not incur a significant costs as long as they are shutdown when not in use. Feel free to research spot instances for further cost reduction. Information on pricing can be found here: <https://aws.amazon.com/ec2/pricing/on-demand/>

root_block_device defines the size and disk type needed for the EC2 Instance. In the example we are deploying 30 GB of disk space on General Purpose SSD Volumes. While servers are turned off Storage Costs are still charged.

encrypted only takes affect when the server is turned off and at rest. This does not protect from data theft while the instance is running.

User Data Configuration

```
user_data = <<EOF
#!/bin/bash
### Standard Patching
sudo dnf --assumeyes update

### Install Firewalld ####
sudo dnf --assumeyes install firewalld
sudo systemctl enable firewalld --now
sudo firewall-cmd --zone=public --permanent --add-service=ssh
sudo firewall-cmd --zone=public --permanent --add-service=http
sudo firewall-cmd --zone=public --permanent --add-service=https
sudo systemctl reload firewalld

### Install Nginx ####
sudo dnf --assumeyes install nginx
sudo systemctl enable nginx --now

### Configure Nginx ####
sudo rm /usr/share/nginx/html/index.html
sudo touch /usr/share/nginx/html/index.html
sudo chown -R nginx:nginx /usr/share/nginx/html

sudo cat > /usr/share/nginx/html/index.html << EOF1
<html>
  <head>
    <title>Welcome to Test Website!</title>
  </head>
  <body>
    <p>It works! Thank you for visiting</b>!</p>
  </body>
</html>
EOF
}
```

[user_data](#) is used for configuring and installing applications during server initial bootup. In this example we update the Linux repositories, patch the server, install a software firewall, configure the firewall to allow in web traffic and ssh, install nginx, and create a static webpage. Further configurations can be accomplished via [AWS Systems Manager](#).

Output AMI Info

```
#####
## Print the Public IP to the Console Screen ##
#####

output "public_ip" { ←
  value = aws_instance.example.public_ip
}
#####
## Print the AMI info out into a variable ##
#####

output "latest_amazon_linux_2_ami_id" { ←
  value = data.aws_ami.latest_amazon_linux_2.id
}
```

Display Server's Public IP after Apply

Display Server's AMI Image After Apply

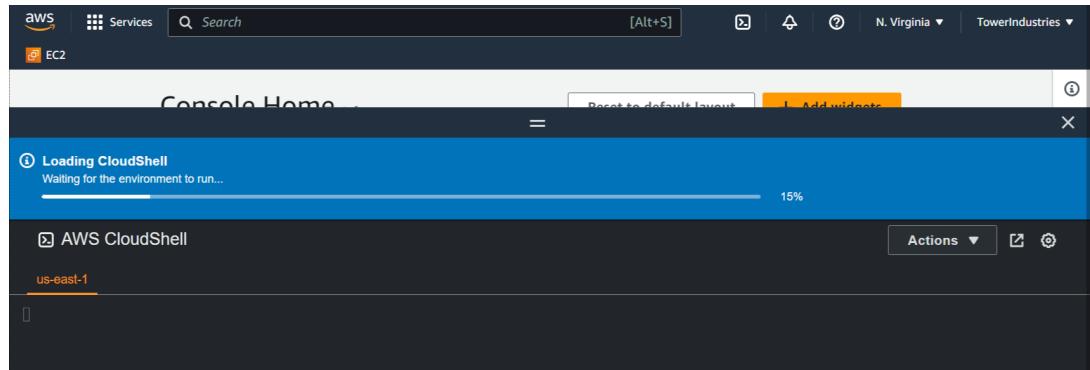
output displays on screen any setting we would like upon applying the terraform plan. In this instance it will display the public IP of the server saving time when configuring Putty for remote access. As part of personal preference the AMI that was chosen using our data resource is also displayed.

Installing Terraform in CloudShell

Step 1: Click “CloudShell Icon”



Step 2: Wait for CloudShell to boot.



[Authentication](#) should be transparent when deploying terraform from the cloud shell. In the event this does not work configure here: [IAM](#) > Users > “Your Username” > Security Credentials > Access Keys > Create Access Key > Command Line Interface > Click “I Understand” box > Next > Key Description > Create Access Key

[Access Key ID](#) is a publicly available identifier that uniquely identifies the AWS account associated with the access key.

[Secret Access Key](#) is a private key used to sign requests made to AWS services.

[AWS configure](#) command will initiate the login process from the cloud shell cli.

Step 3: Type “[mkdir code](#)” > Hit “[Enter](#)”

Step 4: Type “[cd code](#)” > Hit “[Enter](#)”

Step 5: Type “[wget https://releases.hashicorp.com/terraform/1.4.5/terraform_1.4.5_linux_amd64.zip](#)” > Hit “[Enter](#)”

Step 6: Type “[unzip terraform_1.4.5_linux_amd64.zip](#)” > Hit “[Enter](#)”

Step 7: Type “[rm terraform_1.4.5_linux_amd64.zip](#)” > Hit “[Enter](#)”

A screenshot of the AWS CloudShell terminal. The user has run several commands in sequence:

```
[cloudshell-user@ip-10-2-176-217 ~]$ mkdir code
[cloudshell-user@ip-10-2-176-217 ~]$ cd code
[cloudshell-user@ip-10-2-176-217 code]$ wget https://releases.hashicorp.com/terraform/1.4.5/terraform_1.4.5_linux_amd64.zip
--2023-04-21 23:05:55- https://releases.hashicorp.com/terraform/1.4.5/terraform_1.4.5_linux_amd64.zip
Resolving releases.hashicorp.com (releases.hashicorp.com)... 108.138.85.30, 108.138.85.31, 108.138.85.53, ...
Connecting to releases.hashicorp.com (releases.hashicorp.com)|108.138.85.30|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 20767621 (20M) [application/zip]
Saving to: 'terraform_1.4.5_linux_amd64.zip'

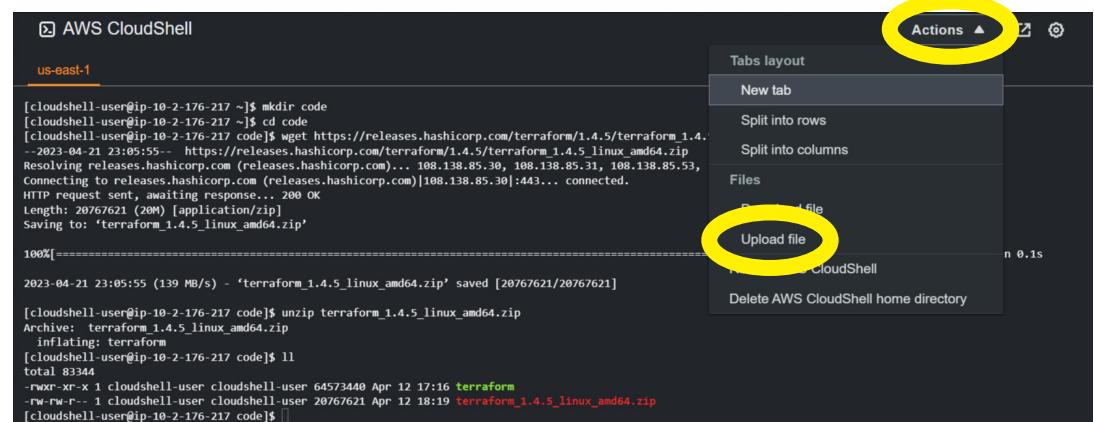
100%[=====] 28,767,621 --.-K/s   in 0.1s
2023-04-21 23:05:55 (139 MB/s) - 'terraform_1.4.5_linux_amd64.zip' saved [20767621/20767621]

[cloudshell-user@ip-10-2-176-217 code]$ unzip terraform_1.4.5_linux_amd64.zip
Archive:  terraform_1.4.5_linux_amd64.zip
      inflating: terraform
[cloudshell-user@ip-10-2-176-217 code]$ ]
```

Upload Terraform.tf

Step 1: Click “**Click Actions**”

Step 2: Click “**Upload file**”



AWS CloudShell (us-east-1)

```
[cloudshell-user@ip-10-2-176-217 ~]$ mkdir code
[cloudshell-user@ip-10-2-176-217 ~]$ cd code
[cloudshell-user@ip-10-2-176-217 code]$ wget https://releases.hashicorp.com/terraform/1.4.5/terraform_1.4.5_linux_amd64.zip
Resolving releases.hashicorp.com (releases.hashicorp.com)... 108.138.85.30, 108.138.85.31, 108.138.85.53
Connecting to releases.hashicorp.com (releases.hashicorp.com)|108.138.85.30|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 20767621 (20M) [application/zip]
Saving to: 'terraform_1.4.5_linux_amd64.zip'

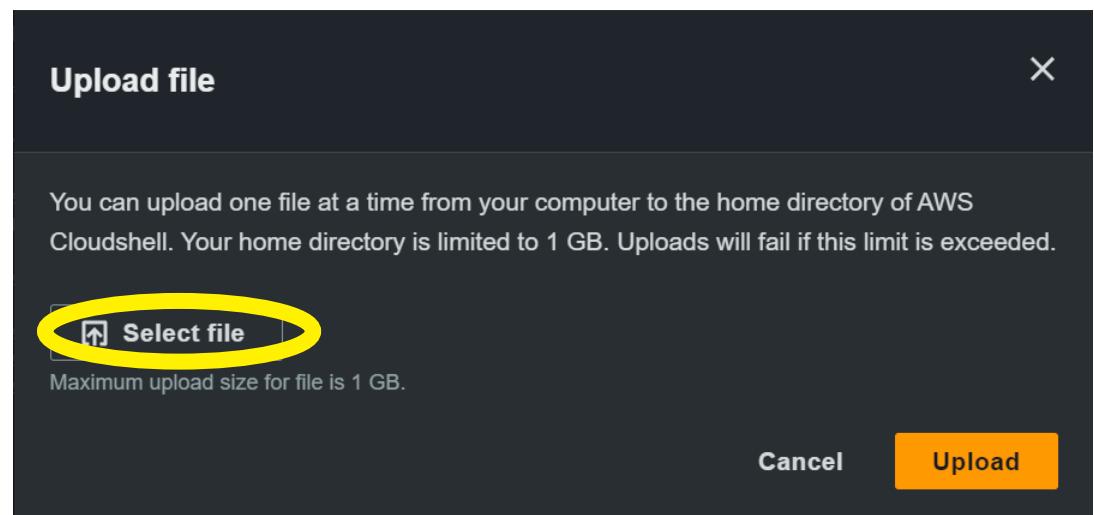
100%[=====] 20767621  (139 MB/s) - 'terraform_1.4.5_linux_amd64.zip' saved [20767621/20767621]

[cloudshell-user@ip-10-2-176-217 code]$ unzip terraform_1.4.5_linux_amd64.zip
Archive:  terraform_1.4.5_linux_amd64.zip
  inflating: terraform
[cloudshell-user@ip-10-2-176-217 code]$ ll
total 83344
-rwxr-xr-x 1 cloudshell-user cloudshell-user 64573440 Apr 12 17:16 terraform
-rw-rw-r-- 1 cloudshell-user cloudshell-user 20767621 Apr 12 18:19 terraform_1.4.5_linux_amd64.zip
[cloudshell-user@ip-10-2-176-217 code]$
```

Actions ▾

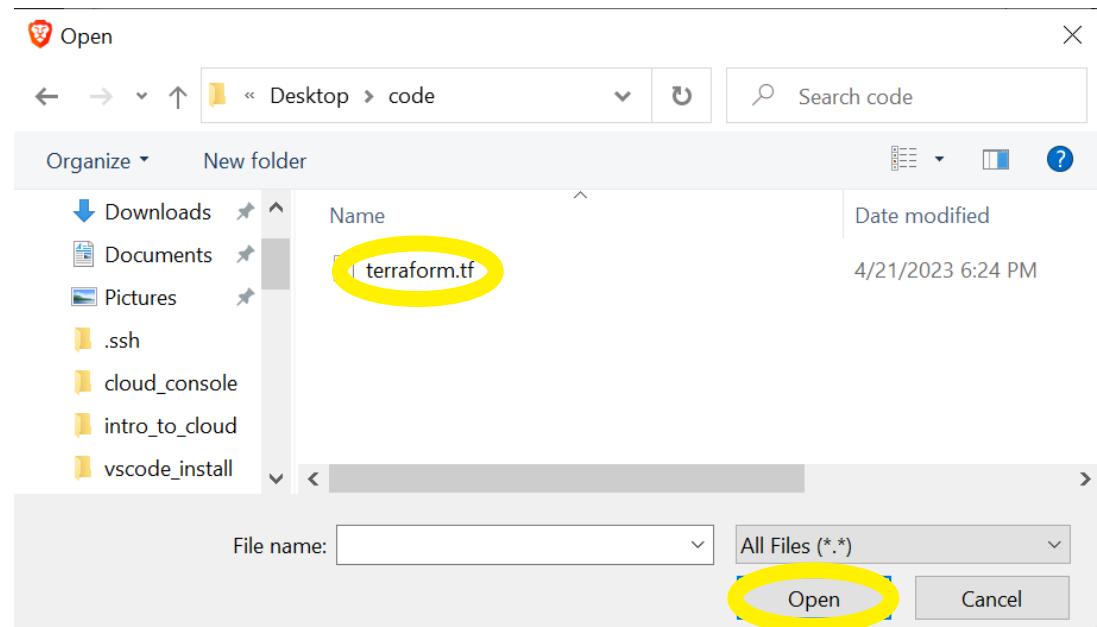
- Tabs layout
- New tab
- Split into rows
- Split into columns
- Files
- Upload file
- Run CloudShell
- Delete AWS CloudShell home directory

Step 3: Click “**Select File**”

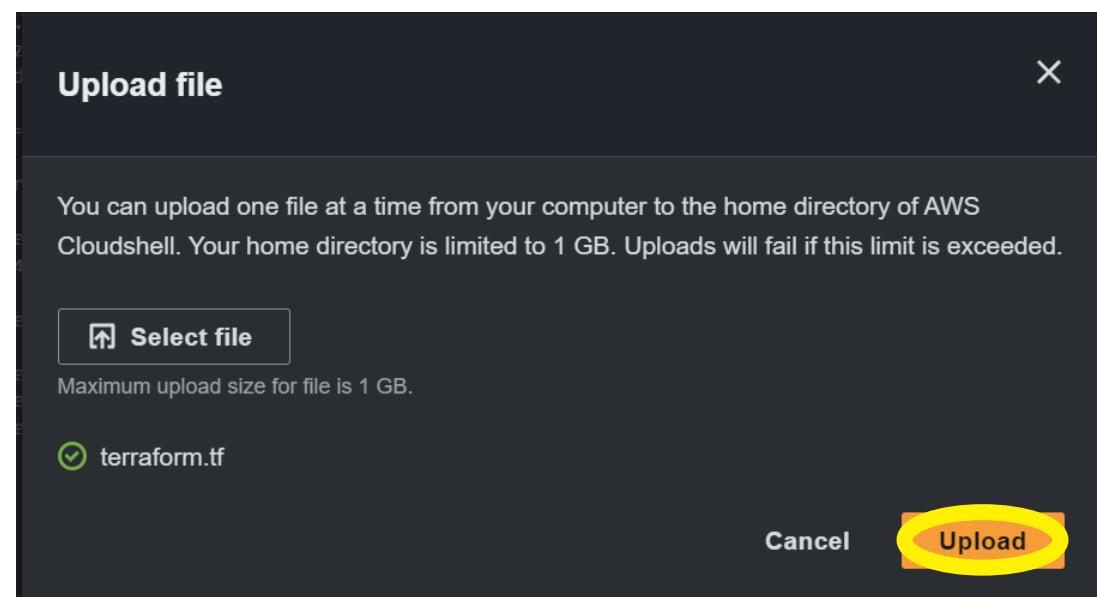


Step 4: Select the terraform file we made “**terraform.tf**”

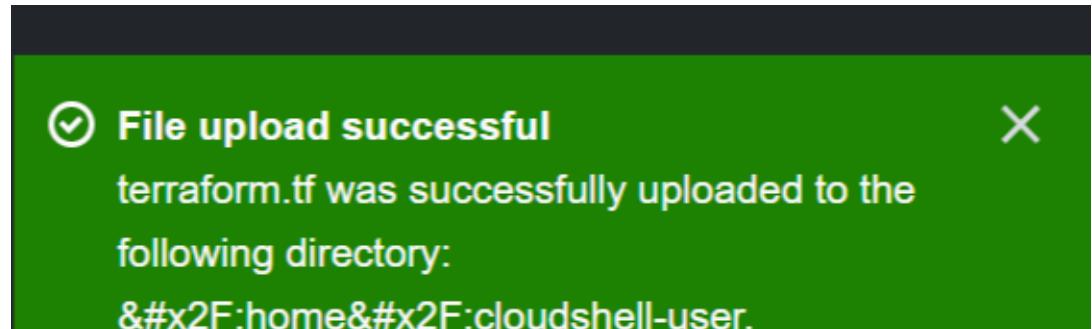
Step 5: Click “**Open**”



Step 6: Click “**Upload**”



Step 7: Verify Successful Upload



Step 7: The file uploads to the root directory not the current one.

Step 8: Type "**cp ..//terraform.tf .**" > Hit "**Enter**"

A screenshot of the AWS CloudShell terminal. The session is named "us-east-1". The terminal shows the following command history:

```
[cloudshell-user@ip-10-2-176-217 code]$ ll  
total 63060  
-rwxr-xr-x 1 cloudshell-user cloudshell-user 64573440 Apr 12 17:16 terraform  
[cloudshell-user@ip-10-2-176-217 code]$ cp ..//terraform.tf .  
[cloudshell-user@ip-10-2-176-217 code]$ ls  
terraform  terraform.tf  
[cloudshell-user@ip-10-2-176-217 code]$ 
```

The "terraform" file is highlighted in green.

Step 9: Type “`./terraform init`” > Hit “Enter”

The screenshot shows a terminal window titled "AWS CloudShell" with the region "us-east-1" selected. The command `./terraform init` is run, and the terminal displays the initialization process. It shows the backend being initialized, provider plugins being found and installed (specifically hashicorp/aws v4.62.0), and a lock file (`.terraform.lock.hcl`) being created to record provider selections. A success message indicates Terraform has been initialized successfully. A note at the bottom suggests rerunning the command if dependencies change.

```
[cloudshell-user@ip-10-2-176-217 code]$ ./terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "4.62.0"...
- Installing hashicorp/aws v4.62.0...
- Installed hashicorp/aws v4.62.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider selections it made above. Include this file in your version control repository so that Terraform can guarantee to make the same selections by default when you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see any changes that are required for your infrastructure. All Terraform commands should now work.

If you ever set or change modules or backend configuration for Terraform, rerun this command to reinitialize your working directory. If you forget, other commands will detect it and remind you to do so if necessary.
[cloudshell-user@ip-10-2-176-217 code]$
```

[terraform init](#) downloads the AWS provider plugin into the working folder under the directory “[.terraform/providers](#)”.

[.terraform.lock.hcl](#) contains information about the current state of the dependencies required by the Terraform configuration, including the versions of the providers, modules, and plugins that were used during the last Terraform run.

When Terraform runs, it checks this file to ensure that the dependencies are up-to-date and that there are no conflicts between different versions of the dependencies. If any changes are made to the configuration, Terraform will update the lock file to reflect the new state of the dependencies.

[modules](#) (not covered here) allow splitting out groups of Terraform configuration constructs (written in the Terraform language) into reusable abstractions.

Step 10: Type “`./terraform plan`” > Hit “Enter”

```
aws CloudShell

us-east-1
+   + "Name" = "dev-security-group"
}
+ tags_all =
+   + "Name" = "dev-security-group"
}
+ vpc_id           = (known after apply)
}

# aws_subnet.example will be created
+ resource "aws_subnet" "example" {
+   arn
+   assign_ipv6_address_on_creation
+   availability_zone
+   availability_zone_id
+   cidr_block
+   enable_dns64
+   enable_resource_name_dns_a_record_on_launch
+   enable_resource_name_dns_aaaa_record_on_launch
+   id
+   ipv6_cidr_block_association_id
+   ipv6_native
+   map_public_ip_on_launch
+   owner_id
+   private_dns_hostname_type_on_launch
+   tags
+     + "Name" = "dev-subnet"
}
+ tags_all =
+   + "Name" = "dev-subnet"
}
+ vpc_id           = (known after apply)

# aws_vpc.example will be created
+ resource "aws_vpc" "example" {
+   arn
+   cidr_block
+   default_network_acl_id
+   default_route_table_id
+   default_security_group_id
+   dhcp_options_id
+   enable_classiclink
+   enable_classiclink_dns_support
+   enable_dns_hostnames
+   enable_dns_support
+   enable_network_address_usage_metrics
+   id
+   instance_tenancy
+   ipv6_association_id
+   ipv6_cidr_block
+   ipv6_cidr_block_network_border_group
+   main_route_table_id
+   owner_id
+   tags
+     + "Name" = "dev-vpc"
}
+ tags_all =
+   + "Name" = "dev-vpc"
}

Plan: 8 to add, 0 to change, 0 to destroy.

Changes to Outputs:
+ latest_amazon_linux_2_ami_id = "ami-02396cdd13e9a1257"
+ public_ip                  = (known after apply)

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
[cloudshell-user@ip-10-2-176-217 code]$
```

In this image we see that there will be 8 changes made to the existing AWS configuration, Zero changes to be made, and Zero to be destroyed. This is the most important thing to look for when applying your code. This is where you will catch any mistakes you may have made.

Step 11: Type “**./terraform apply**” > Hit “**Enter**”

Step 12: Type “**yes**” > Hit “**Enter**”

```
☒ AWS CloudShell

us-east-1

    + "Name" = "dev-subnet"
}
+ vpc_id
}

# aws_vpc.example will be created
+ resource "aws_vpc" "example" {
    + arn
    + cidr_block
    + default_network_acl_id
    + default_route_table_id
    + default_security_group_id
    + dhcp_options_id
    + enable_classiclink
    + enable_classiclink_dns_support
    + enable_dns_hostnames
    + enable_dns_support
    + enable_network_address_usage_metrics
    + id
    + instance_tenancy
    + ipv6_association_id
    + ipv6_cidr_block
    + ipv6_cidr_block_network_border_group
    + main_route_table_id
    + owner_id
    + tags
        + "Name" = "dev-vpc"
    }
    + tags_all
        + "Name" = "dev-vpc"
    }
}

Plan: 8 to add, 0 to change, 0 to destroy.

Changes to Outputs:
+ latest_amazon_linux_2_ami_id = "ami-02396cdd13e9a1257"
+ public_ip = (known after apply)

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

aws_vpc.example: Creating...
aws_vpc.example: Creation complete after 1s [id=vpc-00a180d4fc5fcc78e]
aws_subnet.example: Creating...
aws_internet_gateway.example: Creating...
aws_route_table.example: Creating...
aws_security_group.example: Creating...
aws_internet_gateway.example: Creation complete after 0s [id=igw-0beb7065c29086544]
aws_route_table.example: Creation complete after 0s [id=rtb-00e7537368edf4744]
aws_route.default_route: Creating...
aws_subnet.example: Creation complete after 0s [id=subnet-056d86b8c4bc976c6]
aws_route_table_association.example: Creating...
aws_route.default_route: Creation complete after 1s [id=r-rtb-00e7537368edf47441080289494]
aws_route_table_association.example: Creation complete after 1s [id=rtbassoc-04433e312c1fa2e56]
aws_security_group.example: Creation complete after 2s [id=sg-03b18c6ed4a16bea5]
aws_instance.example: Creating...
aws_instance.example: Still creating... [10s elapsed]
aws_instance.example: Creation complete after 12s [id=i-0a59369e98c1b6fc2]

Apply complete! Resources: 8 added, 0 changed, 0 destroyed.

outputs:

latest amazon linux 2 ami id = "ami-02396cdd13e9a1257"
public ip = "3.87.70.240"
```

During this final Apply verify the Add, Changes, and Destroy numbers.

Outputs allow you to retrieve information from these created or updated resources, such as IP addresses, resource IDs, or DNS names. Here are two things that are of value. Show the AMI that was chosen and the Public IP address of the server for quick Putty configuration.